Silp 1 A

```
/*
Write menu driven program using 'C' for Binary Search Tree. The menu includes
 -       Create a Binary Search Tree
 -       Insert element in a Binary Search Tree
 -       Display
*/

#include <stdio.h>
#include <stdlib.h>


struct btnode
{
    int value;
    struct btnode *l;
    struct btnode *r;
}*root = NULL, *temp = NULL, *t2, *t1;


// Function declaration
void create();
void insert();
void search(struct btnode *t);
void postorder(struct btnode *t);

int flag = 1;

void main() {
    int ch;

    printf("\nOPERATIONS ---");
    printf("\n1 - Insert an element into tree\n");
    printf("2 - Display");
    printf("\n3 - Exit\n");
    while(1) {
        printf("\n\nEnter your choice : ");
        scanf("%d", &ch);

        switch (ch) {
            case 1: insert(root);
                                    break;
            case 2: postorder(root);
                                    break;

            case 3: exit(0);

                                    default : printf("Invalid Input!");
                                     break;
        }
    }
}


void create() {
    int data;

    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
}


void insert() {
    create();
    if (root == NULL)
        root = temp;
    else
        search(root);
}


void search(struct btnode *t) {
    if ((temp->value > t->value) && (t->r != NULL)) {
                /* value more than root node value insert at right */
        search(t->r);
            }
            else if ((temp->value > t->value) && (t->r == NULL)) {
                        t->r = temp;
            }
    else if ((temp->value < t->value) && (t->l != NULL)) {
               /* value less than root node value insert at left */
        search(t->l);
    }
    else if ((temp->value < t->value) && (t->l == NULL)) {
            t->l = temp;
            }
}


void postorder(struct btnode *t) {
    if (root == NULL) {
        printf("No elements in a tree to display ");
        return;
    }

            if (t->l != NULL) {
            postorder(t->l);
            }

    if (t->r != NULL) {
            postorder(t->r);
            }

    printf("%d  ", t->value);
}
```

Silp 1 B

```
/*Write a 'C' program to evaluate a given polynomial using function. (Use array)*/

#include<stdio.h>
#include<math.h>

// function
int evaluate(int arr[], int limit, int x) {
            int sum = 0, count;
    for(count = limit; count >= 0; count--) {
        sum = sum + arr[count]*pow(x, count);
    }
    return sum;
}


int main() {
    int array[30], degree, x, count, result, i;

    // accepting degree
    printf("\nEnter the Degree of Polynomial: ");
    scanf("%d", &degree);

    // accepting co-efficieents
    printf("\nEnter the Co - Efficients:\n");
    for(count = degree; count >= 0; count--) {
            printf("\nCo - Efficient of A[%d]: ", count);
        scanf("%d", &array[count]);
    }

    // logical display
    printf("\nThe Polynomial:\n\n");
    for(i = degree; i >= 0; i--) {
                        if(array[i] != 0) {
            printf("%d^%d + ", array[i], i);
        }
            }

    printf("%d", array[count]);

    printf("\n\nEnter the Value of X: ");
    scanf("%d", &x);

    // function call
    result = evaluate(array, degree, x);
    printf("\nEvaluation of Polynomial: %d\n", result);

    return 0;
}
```

Silp 2 A

```
// Write a 'C' program to accept a string from user and reverse it using Static implementation of Stack

#include <stdio.h>
#include <string.h>

#define MAX 50

// Global variables
int top = -1;
int item;
char stack_string[MAX];
```

```
// function declaration
void pushChar(char item);
char popChar(void);
int isEmpty(void);
int isFull(void);


int main() {
    char str[MAX];

    int i;

    printf("Input a string: ");
    scanf("%[^\n]s",&str);

    for(i=0;i<strlen(str);i++) {
            pushChar(str[i]);
            }

    for(i=0;i<strlen(str);i++) {
        str[i] = popChar();
            }

    printf("Reversed String is: %s\n",str);

    return 0;
}


void pushChar(char item) {
    if(top == MAX-1) {
        printf("\nStack is FULL !!!\n");
        return;
    }

    top = top + 1;
    stack_string[top] = item;
}


char popChar() {
    if(top == -1) {
        printf("\nStack is EMPTY!!!\n");
        return 0;
    }

    item = stack_string[top];
    top = top - 1;
    return item;
}
```

Silp 2B

```
//Write a ‘C’ program to create Circularly Doubly Linked list and display it

#include <stdio.h>
#include <stdlib.h>

struct node {
    int num;
    struct node * nextptr;
}*stnode;


void CIListcreation(int n)
{
    int i, num;
    struct node *preptr, *newnode;

    if(n >= 1)
    {
        stnode = (struct node *)malloc(sizeof(struct node));

        printf(" Input data for node 1 : ");
        scanf("%d", &num);
        stnode->num = num;
        stnode->nextptr = NULL;
        preptr = stnode;
        for(i=2; i<=n; i++)
        {
            newnode = (struct node *)malloc(sizeof(struct node));
            printf(" Input data for node %d : ", i);
            scanf("%d", &num);
            newnode->num = num;
            newnode->nextptr = NULL;    // next address of new node set as NULL
            preptr->nextptr = newnode;  // previous node is linking with new node
            preptr = newnode;           // previous node is advanced
        }
        preptr->nextptr = stnode;                    //last node is linking with first node
    }
}


void displayClList()
{
    struct node *tmp;
    int n = 1;

    if(stnode == NULL)
    {
        printf(" No data found in the List yet.");
    }
    else
    {
        tmp = stnode;
        printf("\n\n Data entered in the list are :\n");

        do {
            printf(" Data %d = %d\n", n, tmp->num);

            tmp = tmp->nextptr;
            n++;
        }while(tmp != stnode);
    }
}


void main()
{
    int n;
    stnode = NULL;
            printf("\n\n Circular Linked List \n");
            printf("--------------------------------------------------------------------\n");

    printf(" Input the number of nodes : ");
    scanf("%d", &n);

    ClListcreation(n);
    displayClList();
}
```

Silp 3A

// Write a program to create two singly linked list of elements of type integer and find the union of the linked lists.

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


struct Node {
        int data;
        struct Node* next;
};


void push(struct Node** head_ref, int new_data);
bool isPresent(struct Node* head, int data);


struct Node* getUnion(struct Node* head1, struct Node* head2) {
        struct Node* result = NULL;
        struct Node *t1 = head1, *t2 = head2;

        while (t1 != NULL) {
                push(&result, t1->data);
                t1 = t1->next;
        }

        while (t2 != NULL) {
                if (!isPresent(result, t2->data))
                push(&result, t2->data);
                t2 = t2->next;
        }

        return result;
}


void push(struct Node** head_ref, int new_data) {
        struct Node* new_node;

        new_node = (struct Node*)malloc(sizeof(struct Node));
```

```
    new_node->data = new_data;

    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}


void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
}


bool isPresent(struct Node* head, int data) {
    struct Node* t = head;

    while (t != NULL) {
        if (t->data == data) {
            return 1;
        }
        t = t->next;
    }

    return 0;
}


int main() {
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;
    struct Node* intersecn = NULL;
    struct Node* unin = NULL;

    // List 1
    push(&head1, 20);
    push(&head1, 4);
    push(&head1, 15);
    push(&head1, 10);

    // List 2
    push(&head2, 10);
    push(&head2, 2);
    push(&head2, 4);
    push(&head2, 8);

    // Create union
    unin = getUnion(head1, head2);


    printf("\n First list is \n");
    printList(head1);

    printf("\n Second list is \n");
    printList(head2);

    printf("\n Union list is \n");
    printList(unin);

    return 0;
}
```

Silp 4A

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


struct btnode
{
   int value;
   struct btnode *l;
   struct btnode *r;
}*root = NULL, *temp = NULL, *t2, *t1;


// Function declaration
void create();
void insert();
void search(struct btnode *t);
void search1(struct btnode *t, int data);
void inorder(struct btnode *t);
void postorder(struct btnode *t);
void delete1(struct btnode *t);


int flag = 1;
```

```
void main() {
    int ch;

    printf("\nOPERATIONS ---");
    printf("\n1 - Insert an element into tree\n");
    printf("2 - Inorder Traversal\n");
    printf("3 - Postorder Traversal\n");
    printf("4 - Exit\n");
    while(1) {
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch (ch) {
        case 1:
            insert();
            break;
        case 2:
            inorder(root);
            break;
        case 3:
            postorder(root);
            break;
        case 4:
            exit(0);
        default :
            printf("Invalid Input!");
            break;
        }
    }
}


void create() {
    int data;

    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
}


void insert() {
    create();
    if (root == NULL)
        root = temp;
    else
        search(root);
}

// find the right position in the tree to insert the data
void search(struct btnode *t) {
    if ((temp->value > t->value) && (t->r != NULL)) {
                /* value more than root node value insert at right */
        search(t->r);
            }
            else if ((temp->value > t->value) && (t->r == NULL)) {
                        t->r = temp;
            }
    else if ((temp->value < t->value) && (t->l != NULL)) {
               /* value less than root node value insert at left */
        search(t->l);
    }
    else if ((temp->value < t->value) && (t->l == NULL)) {
            t->l = temp;
            }
}


// Search for the appropriate position to insert the new node
void search1(struct btnode *t, int data) {
    if ((data>t->value)) {
        t1 = t;
        search1(t->r, data);
    }
    else if ((data < t->value)) {
        t1 = t;
        search1(t->l, data);
    }
    else if ((data==t->value)) {
        delete1(t);
    }
}


void inorder(struct btnode *t) {
    if (root == NULL) {
        printf("No elements in a tree to display");
        return;
    }

    if (t->l != NULL) {
            inorder(t->l);
            }

    printf("%d -> ", t->value);

            if (t->r != NULL) {
                        inorder(t->r);
            }
}


void postorder(struct btnode *t) {
    if (root == NULL) {
        printf("No elements in a tree to display ");
        return;
    }

            if (t->l != NULL) {
            postorder(t->l);
            }

    if (t->r != NULL) {
            postorder(t->r);
            }

    printf("%d -> ", t->value);
}


// To delete a node
void delete1(struct btnode *t) {
    int k;

    // To delete leaf node
    if ((t->l == NULL) && (t->r == NULL)) {
        if (t1->l == t) {
            t1->l = NULL;
        }
        else {
```

```
        t1->r = NULL;
    }
    t = NULL;
    free(t);
    return;
}

// To delete node having one left hand child
else if ((t->r == NULL)) {
    if (t1 == t) {
        root = t->l;
        t1 = root;
    }
    else if (t1->l == t) {
        t1->l = t->l;
                    }
    else {
        t1->r = t->l;
    }
    t = NULL;
    free(t);
    return;
}

// To delete node having right hand child
else if (t->l == NULL) {
    if (t1 == t) {
        root = t->r;
        t1 = root;
    }
    else if (t1->r == t) {
        t1->r = t->r;
                    }
    else {
        t1->l = t->r;
                    }
    t == NULL;
    free(t);
    return;
}

// To delete node having two child
else if ((t->l != NULL) && (t->r != NULL)) {
    t2 = root;
```

```
        if (t->r != NULL) {
            k = smallest(t->r);
            flag = 1;
        }
        else {
            k =largest(t->l);
            flag = 2;
        }
        search1(root, k);
        t->value = k;
    }
}


// To find the smallest element in the right sub tree
int smallest(struct btnode *t) {
    t2 = t;
    if (t->l != NULL) {
        t2 = t;
        return(smallest(t->l));
    }
    else {
            return (t->value);
            }
}


// To find the largest element in the left sub tree
int largest(struct btnode *t) {
    if (t->r != NULL) {
        t2 = t;
        return(largest(t->r));
    }
    else {
             return(t->value);
            }
}
```

Silp 4B

```
/*Write a 'C' program to accept two polynomial and find the addition of accepted
polynomials.(use array)*/

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

```
main() {
	int a[10], b[10], c[10],m,n,k,k1,i,j,x;


	printf("\nPolynomial Addition\n");
	printf("===================\n");



	printf("\nEnter the no. of terms of the polynomial:");
	scanf("%d", &m);


	printf("\nEnter the degrees and coefficients:");
	for (i=0;i<2*m;i++) {
		scanf("%d", &a[i]);
	}

	printf("\nFirst polynomial is:");

	k1=0;
	if(a[k1+1]==1)
	printf("x^%d", a[k1]);
	else
	printf("%dx^%d", a[k1+1],a[k1]);
	k1+=2;

	while (k1<i) {
		printf("+%dx^%d", a[k1+1],a[k1]);
		k1+=2;
	}


	printf("\n\nEnter the no. of terms of 2nd polynomial:");
	scanf("%d", &n);


	printf("\nEnter the degrees and co-efficients:");

	for(j=0;j<2*n;j++) {
		scanf("%d", &b[j]);
	}
	printf("\nSecond polynomial is:");
```

```
k1=0;
if(b[k1+1]==1) {
        printf("x^%d", b[k1]);
}
else {
        printf("%dx^%d",b[k1+1],b[k1]);
}

k1+=2;
while (k1<2*n) {
        printf("+%dx^%d", b[k1+1],b[k1]);
        k1+=2;
}


i=0;
j=0;
k=0;

while (m>0 && n>0) {
        if (a[i]==b[j]) {
                c[k+1]=a[i+1]+b[j+1];
                c[k]=a[i];
                m--;
                n--;
                i+=2;
                j+=2;
        }

        else if (a[i]>b[j]) {
                c[k+1]=a[i+1];
                c[k]=a[i];
                m--;
                i+=2;
        }

        else {
                c[k+1]=b[j+1];
                c[k]=b[j];
                n--;
                j+=2;
        }
```

```
    k+=2;
}

while (m>0) {
    c[k+1]=a[i+1];
    c[k]=a[i];
    k+=2;
    i+=2;
    m--;
}

while (n>0) {
    c[k+1]=b[j+1];
    c[k]=b[j];
    k+=2;
    j+=2;
    n--;
}


printf("\n\nSum of the two polynomials is:");

k1=0;
if (c[k1+1]==1) {
    printf("x^%d", c[k1]);
}
else {
    printf("%dx^%d", c[k1+1],c[k1]);
}

k1+=2;

while (k1<k) {
    if (c[k1+1]==1) {
        printf("+x^%d", c[k1]);
    }
    else {
        printf("+%dx^%d", c[k1+1], c[k1]);
    }
    k1+=2;
}


getch();
```

```
        return 0;

}
```

Silp 5A

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 05A.c
@PritKalariya
PritKalariya Restructuring files.
1 contributor
242 lines (205 sloc) 4.67 KB

```
/*
Write menu driven program using 'C' for Binary Search Tree. The menu includes
  - Create a Binary Search Tree
  -        Traverse it by using Inorder and Preorder traversing technique
*/

#include <stdio.h>
#include <stdlib.h>


struct btnode
{
    int value;
    struct btnode *l;
    struct btnode *r;
}*root = NULL, *temp = NULL, *t2, *t1;
```

```
// Function declaration
void create();
void insert();
void search(struct btnode *t);
void search1(struct btnode *t, int data);
void inorder(struct btnode *t);
void preorder(struct btnode *t);
void delete1(struct btnode *t);


int flag = 1;


void main() {
    int ch;

    printf("\nOPERATIONS ---");
    printf("\n1 - Insert an element into tree\n");
    printf("2 - Preorder Traversal\n");
    printf("3 - Postorder Traversal\n");
    printf("4 - Exit\n");
    while(1) {
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch (ch) {
        case 1:
            insert();
            break;
        case 2:
            inorder(root);
            break;
        case 3:
            preorder(root);
            break;
        case 4:
            exit(0);
        default :
            printf("Invalid Input!");
            break;
        }
    }
}
```

```
void create() {
    int data;

    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
}


void insert() {
    create();
    if (root == NULL)
        root = temp;
    else
        search(root);
}

// find the right position in the tree to insert the data
void search(struct btnode *t) {
    if ((temp->value > t->value) && (t->r != NULL)) {
                /* value more than root node value insert at right */
        search(t->r);
            }
            else if ((temp->value > t->value) && (t->r == NULL)) {
                        t->r = temp;
            }
    else if ((temp->value < t->value) && (t->l != NULL)) {
               /* value less than root node value insert at left */
        search(t->l);
    }
    else if ((temp->value < t->value) && (t->l == NULL)) {
            t->l = temp;
            }
}


// Search for the appropriate position to insert the new node
void search1(struct btnode *t, int data) {
    if ((data>t->value)) {
        t1 = t;
        search1(t->r, data);
    }
    else if ((data < t->value)) {
        t1 = t;
        search1(t->l, data);
    }
    else if ((data==t->value)) {
        delete1(t);
    }
}


void inorder(struct btnode *t) {
    if (root == NULL) {
        printf("No elements in a tree to display");
        return;
    }

    if (t->l != NULL) {
            inorder(t->l);
            }

    printf("%d -> ", t->value);

            if (t->r != NULL) {
                        inorder(t->r);
            }
}


void preorder(struct btnode *t) {
    if (root == NULL) {
        printf("No elements in a tree to display");
        return;
    }
    printf("%d -> ", t->value);

            if (t->l != NULL) {
            preorder(t->l);
            }

            if (t->r != NULL) {
            preorder(t->r);
            }
}
```

```
// To delete a node
void delete1(struct btnode *t) {
    int k;

    // To delete leaf node
    if ((t->l == NULL) && (t->r == NULL)) {
        if (t1->l == t) {
            t1->l = NULL;
        }
        else {
            t1->r = NULL;
        }
        t = NULL;
        free(t);
        return;
    }

    // To delete node having one left hand child
    else if ((t->r == NULL)) {
        if (t1 == t) {
            root = t->l;
            t1 = root;
        }
        else if (t1->l == t) {
            t1->l = t->l;
                        }
        else {
            t1->r = t->l;
        }
        t = NULL;
        free(t);
        return;
    }

    // To delete node having right hand child
    else if (t->l == NULL) {
        if (t1 == t) {
            root = t->r;
            t1 = root;
        }
        else if (t1->r == t) {
            t1->r = t->r;
                        }
        else {
            t1->l = t->r;
                        }
        t == NULL;
        free(t);
        return;
    }

    // To delete node having two child
    else if ((t->l != NULL) && (t->r != NULL)) {
        t2 = root;
        if (t->r != NULL) {
            k = smallest(t->r);
            flag = 1;
        }
        else {
            k =largest(t->l);
            flag = 2;
        }
        search1(root, k);
        t->value = k;
    }
}


// To find the smallest element in the right sub tree
int smallest(struct btnode *t) {
    t2 = t;
    if (t->l != NULL) {
        t2 = t;
        return(smallest(t->l));
    }
    else {
            return (t->value);
            }
}


// To find the largest element in the left sub tree
int largest(struct btnode *t) {
    if (t->r != NULL) {
        t2 = t;
        return(largest(t->r));
    }
    else {
             return(t->value);
            }
}
```

Silp 5B

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 05B.c
@PritKalariya
PritKalariya Restructuring files.
 1 contributor
50 lines (40 sloc) 880 Bytes

```
/*Write a 'C' program to create linked list with given number in which data part of each node contains individual digit of the number.*/

#include<stdio.h>
#include<conio.h>
#include<malloc.h>

struct node {
        int data;
        struct node *next;
};

struct node *start=NULL, *temp=NULL;


int main() {
        int num,a[10],i,j;
```

```
    printf("enter the number: ");
    scanf("%d",&num);

    i=0;
    while(num>0) {
        a[i]=num%10;
        i++;
        num=num/10;
    }

    i--;
    printf("\nthe display of linked list is:-\n");
    for(j=i;j>=0;j--) {
        if(start==NULL) {
            start=(struct node *)malloc(sizeof(struct node));
            start->data=a[j];
            printf("%d",start->data);
            start->next=NULL;
            temp=start;
        }

        else {
            temp->next=(struct node *)malloc(sizeof(struct node));
            temp->next->data=a[j];
            printf(",%d",temp->next->data);
            temp->next->next=NULL;
            temp=temp->next;
        }
    }

    getch();
    return 0;
}
```

Silp 6A

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code

Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 06A.c
@PritKalariya
PritKalariya Restructuring files.
1 contributor
242 lines (205 sloc) 4.69 KB

```
/*
Write menu driven program using 'C' for Binary Search Tree. The menu includes
  -        Create a Binary Search Tree
  -        Traverse it by using Preorder and Postorder traversing technique
*/

#include <stdio.h>
#include <stdlib.h>


struct btnode
{
    int value;
    struct btnode *l;
    struct btnode *r;
}*root = NULL, *temp = NULL, *t2, *t1;


// Function declaration
void create();
void insert();
void search(struct btnode *t);
void search1(struct btnode *t, int data);
void preorder(struct btnode *t);
void postorder(struct btnode *t);
void delete1(struct btnode *t);


int flag = 1;


void main() {
    int ch;

    printf("\nOPERATIONS ---");
    printf("\n1 - Insert an element into tree\n");
    printf("2 - Inorder Traversal\n");
    printf("3 - Postorder Traversal\n");
    printf("4 - Exit\n");
    while(1) {
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch (ch) {
        case 1:
            insert();
            break;
        case 2:
            preorder(root);
            break;
        case 3:
            postorder(root);
            break;
        case 4:
            exit(0);
        default :
            printf("Invalid Input!");
            break;
        }
    }
}


void create() {
    int data;

    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
}


void insert() {
    create();
    if (root == NULL)
```

```
        root = temp;
    else
        search(root);
}

// find the right position in the tree to insert the data
void search(struct btnode *t) {
    if ((temp->value > t->value) && (t->r != NULL)) {
                /* value more than root node value insert at right */
        search(t->r);
            }
            else if ((temp->value > t->value) && (t->r == NULL)) {
                        t->r = temp;
            }
    else if ((temp->value < t->value) && (t->l != NULL)) {
               /* value less than root node value insert at left */
        search(t->l);
    }
    else if ((temp->value < t->value) && (t->l == NULL)) {
            t->l = temp;
            }
}


// Search for the appropriate position to insert the new node
void search1(struct btnode *t, int data) {
    if ((data>t->value)) {
        t1 = t;
        search1(t->r, data);
    }
    else if ((data < t->value)) {
        t1 = t;
        search1(t->l, data);
    }
    else if ((data==t->value)) {
        delete1(t);
    }
}


void preorder(struct btnode *t) {
    if (root == NULL) {
        printf("No elements in a tree to display");
        return;
    }
    printf("%d -> ", t->value);

            if (t->l != NULL) {
            preorder(t->l);
            }

            if (t->r != NULL) {
            preorder(t->r);
            }
}


void postorder(struct btnode *t) {
    if (root == NULL) {
        printf("No elements in a tree to display ");
        return;
    }

            if (t->l != NULL) {
            postorder(t->l);
            }

    if (t->r != NULL) {
            postorder(t->r);
            }

    printf("%d -> ", t->value);
}


// To delete a node
void delete1(struct btnode *t) {
    int k;

    // To delete leaf node
    if ((t->l == NULL) && (t->r == NULL)) {
        if (t1->l == t) {
            t1->l = NULL;
        }
        else {
            t1->r = NULL;
        }
        t = NULL;
        free(t);
        return;
    }

    // To delete node having one left hand child
    else if ((t->r == NULL)) {
        if (t1 == t) {
            root = t->l;
            t1 = root;
        }
        else if (t1->l == t) {
            t1->l = t->l;
                        }
        else {
            t1->r = t->l;
        }
        t = NULL;
        free(t);
        return;
    }

    // To delete node having right hand child
    else if (t->l == NULL) {
        if (t1 == t) {
            root = t->r;
            t1 = root;
        }
        else if (t1->r == t) {
            t1->r = t->r;
                        }
        else {
            t1->l = t->r;
                        }
        t == NULL;
        free(t);
        return;
    }

    // To delete node having two child
    else if ((t->l != NULL) && (t->r != NULL)) {
        t2 = root;
        if (t->r != NULL) {
            k = smallest(t->r);
            flag = 1;
        }
        else {
            k =largest(t->l);
            flag = 2;
        }
        search1(root, k);
        t->value = k;
    }
}


// To find the smallest element in the right sub tree
int smallest(struct btnode *t) {
    t2 = t;
    if (t->l != NULL) {
        t2 = t;
        return(smallest(t->l));
    }
    else {
            return (t->value);
            }
}


// To find the largest element in the left sub tree
int largest(struct btnode *t) {
    if (t->r != NULL) {
        t2 = t;
        return(largest(t->r));
    }
    else {
             return(t->value);
            }
}
```

Silp 6B

Skip to content

Sign up

PritKalariya

/

SY-BBA-CA-Sem-3-Practical-Slips

Public

Code

Issues

1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 06B.c
@PritKalariya
PritKalariya Restructuring files.
1 contributor
51 lines (39 sloc) 899 Bytes

```
/*Write a 'C' program to accept and sort n elements in ascending order by using bubble sort.*/

#include<stdio.h>
#include<conio.h>

int main()
{
    int i,j,temp,a[20],n;
    printf("Enter the size of array: ");
    scanf("%d", &n);

    printf("Enter elements: ");
    for(i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }

    printf("The original array is: \n");
    for (i = 0; i < n; i++)
    {
        printf("\t%d", a[i]);
    }


    //Bubble Sort Logic
    for (i = 0; i < n-1; i++)
    {
        for (j = 0; j < n-1-i; j++)
        {
            if (a[j] > a[j+1])
            {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }

        }

    }

    printf("\n");

    printf("\nThe sorted array is: \n");
    for (i = 0; i < n; i++)
    {
            printf("\t%d", a[i]);
            }

            return 0;

}
```

Silp 7A

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 07A.c
@PritKalariya
PritKalariya Restructuring files.
 1 contributor
242 lines (205 sloc) 4.61 KB

```
/*
Write menu driven program using 'C' for Binary Search Tree. The menu includes
 -      Create a Binary Search Tree
 -      Display
```

```
  -      Delete a given element from Binary Search Tree
*/


#include <stdio.h>
#include <stdlib.h>


struct btnode
{
    int value;
    struct btnode *l;
    struct btnode *r;
}*root = NULL, *temp = NULL, *t2, *t1;


// Function declaration
void create();
void insert();
void search(struct btnode *t);
void search1(struct btnode *t, int data);
void display(struct btnode *t);
void delete();
void delete1(struct btnode *t);


int flag = 1;


void main() {
    int ch;

    printf("\nOPERATIONS ---");
    printf("\n1 - Insert an element into tree\n");
    printf("2 - Display\n");
    printf("3 - Delete\n");
    printf("4 - Exit\n");
    while(1) {
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch (ch) {
        case 1:
            insert();
            break;
        case 2:
            display(root);
            break;
        case 3:
            delete(root);
            break;
        case 4:
            exit(0);
        default :
            printf("Invalid Input!");
            break;
        }
    }
}


void create() {
    int data;

    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
}


void insert() {
    create();
    if (root == NULL)
        root = temp;
    else
        search(root);
}

// find the right position in the tree to insert the data
void search(struct btnode *t) {
    if ((temp->value > t->value) && (t->r != NULL)) {
                /* value more than root node value insert at right */
        search(t->r);
            }
            else if ((temp->value > t->value) && (t->r == NULL)) {
                        t->r = temp;
            }
    else if ((temp->value < t->value) && (t->l != NULL)) {
               /* value less than root node value insert at left */
        search(t->l);
    }
    else if ((temp->value < t->value) && (t->l == NULL)) {
            t->l = temp;
            }
}


// Search for the appropriate position to insert the new node
void search1(struct btnode *t, int data) {
    if ((data>t->value)) {
        t1 = t;
        search1(t->r, data);
    }
    else if ((data < t->value)) {
        t1 = t;
        search1(t->l, data);
    }
    else if ((data==t->value)) {
        delete1(t);
    }
}


void display(struct btnode *t) {
    if (root == NULL) {
        printf("No elements in a tree to display");
        return;
    }
    printf("%d -> ", t->value);

            if (t->l != NULL) {
            display(t->l);
            }

            if (t->r != NULL) {
            display(t->r);
            }
}


void delete() {
    int data;

    if (root == NULL) {
        printf("No elements in a tree to delete");
        return;
    }

    printf("Enter the data to be deleted : ");
    scanf("%d", &data);
    t1 = root;
    t2 = root;
    search1(root, data);
}


// To delete a node
void delete1(struct btnode *t) {
    int k;

    // To delete leaf node
    if ((t->l == NULL) && (t->r == NULL)) {
        if (t1->l == t) {
            t1->l = NULL;
        }
        else {
            t1->r = NULL;
        }
        t = NULL;
        free(t);
        return;
    }

    // To delete node having one left hand child
    else if ((t->r == NULL)) {
        if (t1 == t) {
            root = t->l;
            t1 = root;
        }
        else if (t1->l == t) {
            t1->l = t->l;
                        }
        else {
            t1->r = t->l;
        }
        t = NULL;
        free(t);
        return;
    }

    // To delete node having right hand child
    else if (t->l == NULL) {
        if (t1 == t) {
            root = t->r;
            t1 = root;
        }
        else if (t1->r == t) {
            t1->r = t->r;
                        }
        else {
            t1->l = t->r;
                        }
        t == NULL;
        free(t);
        return;
    }

    // To delete node having two child
    else if ((t->l != NULL) && (t->r != NULL)) {
        t2 = root;
        if (t->r != NULL) {
            k = smallest(t->r);
            flag = 1;
        }
        else {
            k =largest(t->l);
            flag = 2;
        }
        search1(root, k);
        t->value = k;
    }
}


// To find the smallest element in the right sub tree
int smallest(struct btnode *t) {
    t2 = t;
    if (t->l != NULL) {
        t2 = t;
        return(smallest(t->l));
    }
    else {
            return (t->value);
            }
}


// To find the largest element in the left sub tree
int largest(struct btnode *t) {
    if (t->r != NULL) {
        t2 = t;
        return(largest(t->r));
    }
    else {
             return(t->value);
            }
}
```

Silp 7B

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 07B.c
@PritKalariya
PritKalariya Restructuring files.
 1 contributor
96 lines (73 sloc) 1.99 KB

```
/*Write a 'C' program to create a singly linked list and count total number of nodes in it and
display the list and total number of Nodes.*/

#include <stdio.h>
```

```
#include <stdlib.h>


struct node {
    int num;
    struct node *nextptr;
}*stnode;


void createNodeList(int n) {
    struct node *fnNode, *tmp;
    int num, i;

            stnode = (struct node *)malloc(sizeof(struct node));

            if(stnode == NULL) {
        printf(" Memory can not be allocated.");
    }
    else {
        printf(" Input data for node 1 : ");
        scanf("%d", &num);

        stnode-> num = num;
        stnode-> nextptr = NULL;
        tmp = stnode;

        for(i=2; i<=n; i++)
        {
            fnNode = (struct node *)malloc(sizeof(struct node));
            if(fnNode == NULL) {
                printf(" Memory can not allocated.");
                break;
            }
            else {
                printf(" Input data for node %d : ", i);
                scanf(" %d", &num);

                fnNode->num = num;
                fnNode->nextptr = NULL;
                tmp->nextptr = fnNode;
                tmp = tmp->nextptr;
            }
        }
    }
```

```
}


int NodeCount() {
    int ctr = 0;
    struct node *tmp;
    tmp = stnode;

    while(tmp != NULL) {
        ctr++;
        tmp = tmp->nextptr;
    }
    return ctr;
}


void displayList() {
    struct node *tmp;

    if(stnode == NULL) {
        printf(" No data found in the list.");
    }
    else {
        tmp = stnode;
        while(tmp != NULL) {
            printf(" Data = %d\n", tmp->num);
            tmp = tmp->nextptr;
        }
    }
}

int main()
{
    int n,totalNode;

    printf(" Input the number of nodes : ");
    scanf("%d", &n);

    createNodeList(n);

    printf("\n Data entered in the list are : \n");
    displayList();
    totalNode = NodeCount();
    printf("\n Total number of nodes = %d\n", totalNode);

    return 0;
}
```

Silp 8B

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 08B.c
@PritKalariya
PritKalariya Restructuring files.
 1 contributor
56 lines (43 sloc) 885 Bytes

```
/*Write a 'C' program to accept and sort n elements in ascending order by using insertion sort.*/

#include<stdio.h>
#include<conio.h>

int main()
{
	int i,j=0,x,temp,n,a[20];

	printf("Enter the size of array: ");
    scanf("%d", &n);

    printf("Enter elements: ");
    for(i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }
```

```
  printf("The original array is: \n");
  for (i = 0; i < n; i++) {
    printf("\t%d", a[i]);
  }

      printf("\n");
  printf("\n");

  //Insertion Sort Logic
      for(i=0; i<n; i++)
      {
            temp = a[i];
            j = i-1;

            while(j>=0 && a[j]>temp) {
                  a[j+1]=a[j];
                  j--;
            }

            a[j+1] = temp;

            printf("\n");
            printf("\n");
            for(x=0; x<n; x++) {
                  printf("\t%d",a[x]);
            }
      }

      printf("\n");
      printf("\n");

      printf("The sorted array is: \n");
  for (i = 0; i < n; i++)
  {
      printf("\t%d", a[i]);
      }

      return 0;
}
```

Silp 9B
Skip to content
Sign up

PritKalariya

/

SY-BBA-CA-Sem-3-Practical-Slips

Public

Code

Issues

1

Pull requests

Actions

Projects

Security

Insights

SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 09B.c

@PritKalariya

PritKalariya Restructuring files.

1 contributor

108 lines (82 sloc) 2.27 KB

```
/*Write a 'C' program to create a singly linked list, reverse it and display both the list.*/

#include <stdio.h>
#include <stdlib.h>

struct node {
    int num;
    struct node *nextptr;
}*stnode;


void createNodeList(int n)
{
    struct node *fnNode, *tmp;
    int num, i;

    stnode = (struct node *)malloc(sizeof(struct node));

    if(stnode == NULL) {
        printf(" Memory can not be allocated.");
    }
    else {
        printf(" Input data for node 1 : ");
        scanf("%d", &num);

        stnode-> num = num;
        stnode-> nextptr = NULL;
        tmp = stnode;

        for(i=2; i<=n; i++) {
            fnNode = (struct node *)malloc(sizeof(struct node));

            if(fnNode == NULL) {
                printf(" Memory can not be allocated.");
                break;
            }
            else {
                printf(" Input data for node %d : ", i);
                scanf(" %d", &num);

                fnNode->num = num;
                fnNode->nextptr = NULL;
                tmp->nextptr = fnNode;
                tmp = tmp->nextptr;
            }
        }
    }
}


void reverseDispList()
{
    struct node *prevNode, *curNode;

    if(stnode != NULL) {
        prevNode = stnode;
        curNode = stnode->nextptr;
        stnode = stnode->nextptr;

        prevNode->nextptr = NULL; //convert the first node as last

        while(stnode != NULL) {
            stnode = stnode->nextptr;
            curNode->nextptr = prevNode;

            prevNode = curNode;
            curNode = stnode;
        }
        stnode = prevNode; //convert the last node as head
    }
}
```

```
void displayList()
{
    struct node *tmp;

    if(stnode == NULL) {
        printf(" No data found in the list.");
    }
    else {
        tmp = stnode;

        while(tmp != NULL) {
            printf(" Data = %d\n", tmp->num);
            tmp = tmp->nextptr;
        }
    }
}

int main()
{
    int n;

    printf(" Input the number of nodes : ");
    scanf("%d", &n);
    createNodeList(n);

    printf("\n Data entered in the list are : \n");
    displayList();

    reverseDispList();

    printf("\n The list in reverse are :  \n");
    displayList();

    return 0;
}
```

Silp 11 A

Skip to content
Sign up
PritKalariya
/

SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 11A.c
@PritKalariya
PritKalariya Changed extension to .c
1 contributor
152 lines (116 sloc) 2.26 KB

```
/*Write a menu driven program using 'C' for singly linked list-
-         To create linked list.
-         To display linked list
-         To search node in linked list.
-         Insert at last position
*/

#include<stdio.h>
#include<stdlib.h>
#include<conio.h>

struct node {
          int data;
          struct node *link;
}*start;


// 1. create
void create(int data) {
          struct node *q,*tmp;

          tmp=(struct node *)malloc(sizeof(struct node));

          tmp->data=data;
          tmp->link=NULL;

          if(start==NULL)          {
                    start=tmp;
          }
```

```
    else {
        q=start;

        while(q->link!=NULL) {
            q=q->link;
        }

        q->link=tmp;
    }
}


// 2. Display
void display() {
    struct node *q;

    if(start==NULL) {
        printf("\nLIST IS EMPTY");
    }
    else {
        q=start;

        while(q!=NULL) {
            printf("%d->",q->data);
            q=q->link;
        }

        printf("NULL");
    }
}


// 3. Search
void search(int data) {
    struct node *q,*tmp;
    q=start;

    while(q!=NULL) {
        if(q->data==data) {
            printf("\nElement Is Found");
            break;
        }
        else {
            q=q->link;
```

```
                        }
            }

            if(q==NULL) {
                        printf("\nElement is Not Found");
            }
}


// 4. Insert at last
void insert(int data) {
    struct node *newNode, *temp;

    newNode = (struct node*)malloc(sizeof(struct node));

    if(newNode == NULL) {
        printf("Unable to allocate memory :(");
    }
    else {
        newNode->data = data;
        newNode->link = NULL;

        temp = data;

        while(temp != NULL && temp->link != NULL) {
            temp = temp->link;
                        }

        temp->link = newNode;

        printf("Data inserted succesfully :) ");
    }
}


int main() {
            int ch,n,i,m,a,pos;
            start=NULL;

            do {
                        printf("MENU");
                        printf("\n1.Create");
                        printf("\n2.Display");
                        printf("\n3.Search");
                        printf("\n4.Insert at last");
                        printf("\n5.Exit");
                        printf("\n\nEnter your choice: ");
                        scanf("%d",&ch);

                        switch(ch) {
                            case 1:printf("\n\nHow many nodes do you want to create? ");
                                    scanf("%d",&n);
                                    for(i=0;i<n;i++) {
                                        printf("\nEnter the data: ");
                                        scanf("%d",&m);
                                        create(m);
                                    }
                                    break;

                            case 2: display();
                                    break;

                            case 3: printf("\nEnter the element for search: ");
                                    scanf("%d",&m);
                                    search(m);
                                    break;

                            case 4: printf("\nEnter the data: ");
                                    scanf("%d",&m);
                                    insert(m);

                            case 5: exit(0);
                        }

            } while(ch!=7);

            getch();
            return 0;
}
```

Silp 11B

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips

Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 11B.c
@PritKalariya
PritKalariya DS Practical Slip (11B, 13B).
1 contributor
81 lines (66 sloc) 1.19 KB

```
/* Write a menu driven program using 'C' for Dynamic implementation of Queue for integers. The
menu includes
-       Insert
-       Delete
-       Display
-       Exit
*/

#include<stdio.h>

int queue[];
int front = 0;
int rear = 0;

void main() {
    int ch;

    while(1) {
        printf("\nMENU\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch(ch) {
            case 1: insert();
                    break;
            case 2: del();
                    break;

            case 3: display();
                    break;

            case 4: exit(1);

            default: printf("\nInvalid Input!!\n");
        }
    }
}

void insert() {
    int data;
    printf("Enter data: ");
    scanf("%d", &data);

    queue[rear] = data;
    rear++;

    printf("\nData entered successfully!!\n");
}

void del() {
    int i;
    if(front == rear) {
        printf("\nThe queue is Empty!!\n");
    }
    else {
        printf("\n%d deleted\n", queue[front]);
        for(i=0; i<rear-1; i++) {
            queue[i] = queue[i + 1];
        }
        rear--;
    }
}

void display() {
    int i;
    if(front == rear) {
        printf("\nThe queue is Empty!!\n");
    }
```

```
        else {
                printf("\nThe queue elements are: ");
                for(i=front; i<rear; i++) {
                        printf("%d\t", queue[i]);
                }
                printf("\n");
        }
}
```

Silp 12B

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 12B.c
@PritKalariya
PritKalariya Changed extension to .c
 1 contributor
62 lines (52 sloc) 1020 Bytes

```
/*Write a 'C' program to accept and sort n elements in ascending order using Selection sort
method.*/

#include<stdio.h>
#include<conio.h>

int main()
{
        int i,j,temp,min,n,a[20], x;
        printf("Enter the size of array: ");
    scanf("%d", &n);

    printf("Enter elements: ");
    for(i = 0; i < n; i++)
```

```
{
   scanf("%d", &a[i]);
}

printf("The original array is: \n");
for (i = 0; i < n; i++)
{
   printf("\t%d", a[i]);
}
printf("\n");
printf("\n");

//Selction Sort Logic
      for(i=0; i<n-1; i++)
      {
               min = i;
               for(j=i+1; j<n; j++)
               {
                        if(a[j] < a[min])
                        {
                                 //storing the index value of the smallest element
                                 min = j;
                        }
               }

               //swaping the minimum values
               temp = a[i];
               a[i] = a[min];
               a[min] = temp;

               printf("\n");
               printf("\n");
               for(x = 0; x < n; x++)
               {
                        printf("\t%d",a[x]);
               }
      }

      printf("\n");
      printf("\n");

      printf("The sorted array is: \n");
for (i = 0; i < n; i++)
{
        printf("\t%d", a[i]);
        }

        return 0;
}
```

Silp 13A

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 13A.c
@PritKalariya
PritKalariya C Practical Slips programs.
 1 contributor
145 lines (119 sloc) 2.45 KB

```
/*
Write a C program to accept an infix expression and convert it into postfix form.(Use Static
Implementation of Stack)
Example: - A * B + C as AB*C+
*/

#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>

#define SIZE 100

char stack[SIZE];
int top = -1;
```

```
void push(char item) {
  if(top >= SIZE-1) {
    printf("\nStack Overflow.");
  }
  else {
    top = top+1;
    stack[top] = item;
  }
}


char pop() {
  char item ;

  if(top <0) {
    printf("stack under flow: invalid infix expression");
    getchar();
    exit(1);
  }
  else {
    item = stack[top];
    top = top-1;
    return(item);
  }
}


int is_operator(char symbol) {
  if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol =='-') {
    return 1;
  }
  else {
  return 0;
  }
}


int precedence(char symbol) {
  if(symbol == '^') {
    return(3);
  }
  else if(symbol == '*' || symbol == '/') {
    return(2);
  }
  else if(symbol == '+' || symbol == '-') {
    return(1);
  }
  else {
    return(0);
  }
}


void InfixToPostfix(char infix_exp[], char postfix_exp[]) {
  int i, j;
  char item;
  char x;

  push('(');
  strcat(infix_exp,")");

  i=0;
  j=0;
  item=infix_exp[i];

  while(item != '\0') {
    if(item == '(') {
      push(item);
    }
    else if( isdigit(item) || isalpha(item)) {
      postfix_exp[j] = item;
      j++;
    }
    else if(is_operator(item) == 1) {
      x=pop();
      while(is_operator(x) == 1 && precedence(x)>= precedence(item)) {
        postfix_exp[j] = x;
        j++;
        x = pop();
      }
      push(x);

      push(item);
    }
    else if(item == ')') {
      x = pop();
      while(x != '(') {
        postfix_exp[j] = x;
        j++;
        x = pop();
      }
    }
    else {
      printf("\nInvalid infix Expression.\n");
      getchar();
      exit(1);
    }
    i++;


    item = infix_exp[i];
  }
  if(top>0) {
    printf("\nInvalid infix Expression.\n");
    getchar();
    exit(1);
  }
  if(top>0) {
    printf("\nInvalid infix Expression.\n");
    getchar();
    exit(1);
  }

  postfix_exp[j] = '\0';

}

int main() {
  char infix[SIZE], postfix[SIZE];

  printf("\nEnter Infix expression : ");
  gets(infix);

  InfixToPostfix(infix,postfix);
  printf("Postfix Expression: ");
  puts(postfix);

  return 0;
}
```

Silp 13B

SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 13B.c
@PritKalariya
PritKalariya DS Practical Slip (11B, 13B).
1 contributor
87 lines (66 sloc) 1.25 KB

```
//Write a 'C' program to create doubly link list and display nodes having odd value

#include<stdio.h>
#include<stdlib.h>

struct node{
    struct node *left;
    int data;
    struct node *right;
};

struct node *root = NULL;

void main() {
    int ch;

    while(1) {
        printf("\nMENU\n");
        printf("1. Append.\n");
        printf("2. Display.\n");
        printf("3. Exit.\n");

        printf("\nEnter you choice: ");
        scanf("%d", &ch);
        switch(ch) {
            case 1: append();
                    break;

            case 2: display();
                    break;

            case 3: exit(0);

            default: printf("\nINVALID INPUT!!\n");
        }
    }
}

// Case 1
void append() {
    struct node *temp;

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

    printf("Enter node data: ");
    scanf("%d", &temp->data);

    temp->left = NULL;
    temp->right = NULL;

    if(root == NULL) {
        root = temp;
    }
    else {
        struct node *p;

        p = root;

        while(p->right != NULL) {
            p = p->right;
        }

        p->right = temp;
        temp->left = p;
    }

    printf("\nData entered successfully.\n");
```

```
}

// Case 2
void display() {
    struct node *temp = root;

    if(temp == NULL) {
        printf("\nTHE LIST IS EMPTY!!\n");
    }
    else {
        while(temp != NULL) {
            if(temp->data % 2 != 0) {
                printf("%d\t", temp->data);
            }
            temp = temp->right;
        }
        printf("\n");
    }
}
```

Silp 14A

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 14A.c
@PritKalariya
PritKalariya C Practical Slips programs.
 1 contributor
78 lines (54 sloc) 1.25 KB

```
// Write a 'C' program to accept a string from user and reverse it using Dynamic implementation of Stack
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <limits.h>


struct Stack {
        int top;
        unsigned capacity;
        char* array;
};


struct Stack* createStack(unsigned capacity) {
        struct Stack* stack = (struct Stack*) malloc(sizeof(struct Stack));
        stack->capacity = capacity;
        stack->top = -1;
        stack->array = (char*) malloc(stack->capacity * sizeof(char));

        return stack;
}


int isFull(struct Stack* stack) {
        return stack->top == stack->capacity - 1;
}


int isEmpty(struct Stack* stack) {
        return stack->top == -1;
}


// Add item
void push(struct Stack* stack, char item) {
        if (isFull(stack)) {
                return;
        }
        stack->array[++stack->top] = item;
}


// remove item
```

```
char pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        return INT_MIN;
    }

    return stack->array[stack->top--];
}


// A stack based function to reverse a string
void reverse(char str[]) {
    int n = strlen(str), i;
    struct Stack* stack = createStack(n);

    for (i = 0; i < n; i++) {
        push(stack, str[i]);
    }

    for (i = 0; i < n; i++) {
        str[i] = pop(stack);
    }
}


int main() {
    char str[] = "TESTING";

    reverse(str);
    printf("Reversed string is %s", str);

    return 0;
}
```

Silp 15B

Skip to content

Sign up

PritKalariya

/

SY-BBA-CA-Sem-3-Practical-Slips

Public

Code

Issues

1

Pull requests

Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 15B.c
@PritKalariya
PritKalariya DS Practical Slip (15,16,18,19).
 1 contributor
90 lines (69 sloc) 1.35 KB

```
/*Write menu driven program using 'C' for Dynamic implementation of Stack. The menu includes
following operations:
-       Push
-       Pop
-       Display
-       Exit
*/

#include<stdio.h>
#include<stdlib.h>

struct node {
        int data;
        struct node *next;
};

struct node *top = NULL;

int main() {

        int ch, n;

        while(1) {
                printf("\nMenu\n");
                printf("1. Push\n");
                printf("2. Pop\n");
                printf("3. Display\n");
                printf("4. Exit\n");

                printf("Enter your choice: ");
                scanf("%d", &ch);

                switch(ch) {
                        case 1: printf("Enter the number: ");
                                        scanf("%d", &n);
```

```
                    push(n);
                    break;

            case 2: del();
                    break;

            case 3: traverse();
                    break;

            case 4: exit(0);

            default: printf("\nINVALID INPUT!\n");
        }
    }

    return 0;
}

// 1. Adding new value
void push(int item) {
    struct node *nptr;

    nptr = (struct node *)malloc(sizeof(struct node));

    nptr->data = item;
    nptr->next = top;
    top = nptr;

    printf("\n%d entered successfully.\n", nptr->data);
}

// 2. Deleting the last entered value
void del() {
    if(top == NULL) {
        printf("\nSTACK IS EMPTY!!\n");
    }
    else {
        struct node *temp;
        temp = top;
        top = top->next;
        printf("\n%d removed\n", temp->data);
        free(temp);
    }
}
```

```
// 3. Display the whole stack
void traverse() {
        struct node *temp;

        temp = top;

        while(temp != NULL) {
                printf("\n%d\t\n", temp->data);
                temp = temp->next;
        }
}
```

Silp 16A

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 16A.c
@PritKalariya
PritKalariya C Practical Slips programs.
 1 contributor
44 lines (32 sloc) 721 Bytes

```
/*
Write a 'C' program which accept the string and reverse each word of the string using Static implementation of stack.
Example: Input - This is an input string
  Output - sihTsinatupnignirts
*/


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

```
#define max 100

int top,stack[max];


void push(char x) {
      if(top == max-1){
        printf("stack overflow");
      }
              else {
        stack[++top]=x;
      }

}

void pop() {
    printf("%c",stack[top--]);
}


main() {
            char str[]="Testing";
            int len = strlen(str);
            int i;

            for(i=0;i<len;i++) {
                        push(str[i]);
            }

            printf("\nThis the reversed string: ");
            for(i=0;i<len;i++) {
                        pop();
            }
}
```

Silp 16B

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public

Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 16B.c
@PritKalariya
PritKalariya DS Practical Slip (15,16,18,19).
 1 contributor
96 lines (73 sloc) 1.47 KB

```
/*Write a 'C' program to create to a Singly linked list. Accept the number from user, search the
number in the list.
- If the number is present display the Position of node.
- If number not present print the message "Number not Found".
*/

#include<stdio.h>
#include<stdlib.h>

struct node {
    int data;
    struct node *link;
};

struct node *root = NULL;

void main() {
    int ch, count;

    while(1) {
        printf("\nMenu\n");
        printf("1. Append\n");
        printf("2. Search\n");
        printf("3. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch(ch) {
            case 1: append();
                    break;
            case 2: count = search();
                    if(count != 0) {
                        printf("\nIndex number: %d\n", count);
                    }
                    else {
                        printf("\nNumber not found.\n");
                    }
                    break;

            case 3: exit(1);

            default: printf("\nINVALID INPUT!!\n");
        }
    }
}

// 1. Append
void append() {
    struct node *temp;

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

    printf("Enter node data: ");
    scanf("%d", &temp->data);
    temp->link = NULL;

    if(root == NULL) {
        root = temp;
    }
    else {
        struct node* p;

        p = root;

        while(p->link != NULL) {
            p = p->link;
        }

        p->link = temp;
    }

    printf("\nData entered successfully.\n");
}
```

```
int search() {
    struct node *p;
    int num, count = 1;

    printf("Enter number you want to search: ");
    scanf("%d", &num);

    p = root;

    while(p != NULL) {
        if(p->data == num) {
            return count;
        }
        else {
            p = p->link;
            count++;
        }
    }

    return 0;
}
```

Silp 17 A

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 17A.c
@PritKalariya
PritKalariya C Practical Slips programs.
 1 contributor
54 lines (42 sloc) 1.04 KB

// Write a 'C' program to read a postfix expression, evaluate it and display the result. (Use Static Implementation of Stack).

```
#include<stdio.h>
int stack[20];
int top = -1;

void push(int x) {
   stack[++top] = x;
}

int pop() {
   return stack[top--];
}

int main() {
   char exp[20];
   char *e;
   int n1,n2,n3,num;

         printf("Enter the expression :: ");
   scanf("%s",exp);

         e = exp;

         while(*e != '\0') {
      if(isdigit(*e)) {
         num = *e - 48;
         push(num);
      }
      else {
         n1 = pop();
         n2 = pop();

                           switch(*e) {
                  case '+': n3 = n1 + n2;
                             break;

                  case '-': n3 = n2 - n1;
                             break;

                  case '*': n3 = n1 * n2;
                          break;
```

```
                        case '/': n3 = n2 / n1;
                                  break;
            }
            push(n3);
        }
        e++;
    }
    printf("\nThe result of expression %s  =  %d\n\n",exp,pop());
    return 0;
}
```

Footer

Footer navigation
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About
SY-BBA-CA-Sem-3-Practical-Slips/Practical Slip 17A.c at main · PritKalariya/SY-BBA-CA-Sem-3-Practical-Slips · GitHub

Silp 18B
Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 18B.c

@PritKalariya
PritKalariya DS Practical Slip (15,16,18,19).
 1 contributor
115 lines (93 sloc) 2.22 KB

```
//Write a 'C' program to accept and sort n elements in ascending order using Merge sort
method.

#include<stdio.h>
#include<conio.h>

void merge(int arr[20], int first, int mid, int last)
{
    int i, j, k;

    //Storing the number of values from first half of the array in n1
    int n1 = (mid - first) + 1;

    //Storing the number of values from second half of the array in n2
    int n2 = last - mid;

    //Declaring to empty arrays
    int l[10], r[10];

    for(i = 0; i < n1; i++)
    {
        //Transfering values of original array to l(only the values of first half)
        l[i] = arr[first + i];
    }

    for(j = 0; j < n2; j++)
    {
        r[j] = arr[mid + 1 + j];
    }

    i = 0;
    j = 0;
    k = first;

    while(i < n1 && j < n2)
    {
        //Comparing the elements from temp. array(l & r) and then storing them to the original array
        if(l[i] <= r[j])
        {
            arr[k] = l[i];
            i++;
        }
        else
        {
            arr[k] = r[j];
            j++;
        }

        k++;
    }

    while(i < n1)
    {
        arr[k] = l[i];
        i++;
        k++;
    }

    while(j < n2)
    {
        arr[k] = r[j];
        j++;
        k++;
    }
}

void mergesort(int arr[20], int first, int last)
{
    int mid;
    if(first < last)
    {
        mid = (first + last) / 2;

        //Divide the first part of the array
        mergesort(arr, first, mid);

        //Divide the second part of the array
        mergesort(arr, mid + 1, last);

        //Merge all the arrays to one
        merge(arr, first, mid, last);
    }
}
```

```
int main()
{
    int arr[20], n, i;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    //Accepting rray elements
    printf("\nEnter the elements: ");
    for(i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    //Printing the original array
    printf("\nThe original array is: \n");
    for(i = 0; i < n; i++)
    {
        printf("\t%d", arr[i]);
    }

    //Calling mergesort function to divide the array into parts
    mergesort(arr, 0, n-1);

    //Printing the sorted array
    printf("\n\nThe sorted array is: \n");
    for(i = 0; i < n; i++)
    {
        printf("\t%d", arr[i]);
    }

    return 0;
}
```

Blog
About
SY-BBA-CA-Sem-3-Practical-Slips/Practical Slip 18B.c at main · PritKalariya/SY-BBA-CA-Sem-3-Practical-Slips · GitHub

Silp 19A

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 19A.c
@PritKalariya
PritKalariya C Practical Slips programs.
 1 contributor
43 lines (32 sloc) 724 Bytes

```
/*
Write a 'C' program which accept the string and reverse each word of the string using Dynamic implementation of stack.
  Example: Input - This is an input string
  Output - sihTsinatupnignirts
*/

#include <stdio.h>
#include <string.h>

#define max 100

int top,stack[max];


void push(char x) {
    if(top == max-1){
        printf("stack overflow");
      }
            else {
        stack[++top]=x;
      }

}

void pop() {
    printf("%c",stack[top--]);
}


main() {
            char str[]="Testing";
            int len = strlen(str);
            int i;

            for(i=0;i<len;i++) {
                        push(str[i]);
            }

            printf("\nThis the reversed string: ");
            for(i=0;i<len;i++) {
                        pop();
            }
}
```

SY-BBA-CA-Sem-3-Practical-Slips/Practical Slip 19A.c at main · PritKalariya/SY-BBA-CA-Sem-3-Practical-Slips · GitHub

Silp 19B
Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 19B.c
@PritKalariya
PritKalariya DS Practical Slip (15,16,18,19).
 1 contributor
85 lines (65 sloc) 1.25 KB

```
//Write a 'C' program to create a singly Link list and display its alternative nodes. (start displaying from first node)

#include<stdio.h>
#include<stdlib.h>

struct node {
    int data;
    struct node *link;
};

struct node *root = NULL;

void main() {
    int ch;

    while(1) {
        printf("\nMenu\n");
        printf("1. Append\n");
        printf("2. Display\n");
        printf("3. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &ch);
        switch(ch) {
            case 1: append();
                    break;

            case 2: display();
                    break;

            case 3: exit(1);

            default: printf("\nINVALID INPUT!!\n");
        }
    }
}

void append() {
    struct node *temp;

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

    printf("Enter node data: ");
    scanf("%d", &temp->data);
    temp->link = NULL;

    if(root == NULL) {
        root = temp;
    }
    else {
        struct node* p;

        p = root;

        while(p->link != NULL) {
            p = p->link;
        }

        p->link = temp;
    }

    printf("\nData entered successfully.\n");
}

void display() {
    int counter = 0;
    struct node* temp;
    temp = root;

    if(temp == NULL) {
        printf("\nLIST IS EMPTY!!\n");
    }
    else {
        printf("\n");
        while(temp != NULL) {
            if(counter % 2 == 0) {
                printf("%d\t", temp->data);
            }
            counter++;
            temp = temp->link;
        }
        printf("\n");
    }
}
```

Footer

Footer navigation
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About
SY-BBA-CA-Sem-3-Practical-Slips/Practical Slip 19B.c at main · PritKalariya/SY-BBA-CA-Sem-3-Practical-Slips · GitHub

Silp 20B
Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues

1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 20B.c
@PritKalariya
PritKalariya DS Practical Slip (20B)
 1 contributor
122 lines (94 sloc) 1.82 KB

```
//Write a 'C' program to swap mth and nth  element of singly linked list.

#include<stdio.h>
#include<conio.h>

struct node{
        int data;
        struct node *link;
};

struct node *root = NULL;

void main() {
        int ch;

        while(1) {
                printf("\nMENU\n");
                printf("1. Append\n");
                printf("2. Swap\n");
                printf("3. Display\n");
                printf("4. Exit\n");

                printf("Enter your choice: ");
                scanf("%d", &ch);

                switch(ch) {
                        case 1: append();
                                        break;

                        case 2: swap();
                                        break;

                        case 3: display();
```

```
                    break;

            case 4: exit(1);

            default: printf("\nINVALID INPUT!!\n");
        }
    }
}

// case 1
void append() {
    struct node *temp;

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

    printf("Enter node data: ");
    scanf("%d", &temp->data);
    temp->link = NULL;

    if(root == NULL) {
        root = temp;
    }
    else {
        struct node* p;

        p = root;

        while(p->link != NULL) {
            p = p->link;
        }

        p->link = temp;
    }

    printf("\nData entered successfully.\n");
}

// Case 2
void swap() {
    struct node *p, *q;
    int m, n, i, temp;

    printf("Enter the Mth position of the node you want to swap: ");
    scanf("%d", &m);
    printf("Enter the Nth position of the node you want to swap with: ");
    scanf("%d", &n);

    p = q = root;

    // Travelling till location m
    for(i = 1; i < m && p != NULL; i++) {
        p = p->link;
    }

    // Travelling till location n
    for(i = 1; i < n && q != NULL; i++) {
        q = q->link;
    }

    // swaping
    if(p != NULL && q != NULL) {
        temp = p->data;
        p->data = q->data;
        q->data = temp;

        printf("\nSwaping successfull.\n");
    }
    else {
        printf("\nINVALID INPUT!!\n");
    }
}

// Case 3
void display() {
    struct node* temp;
    temp = root;

    if(temp == NULL) {
        printf("\nLIST IS EMPTY!!\n");
    }
    else {
        printf("\n");
        while(temp != NULL) {
            printf("%d\t", temp->data);
            temp = temp->link;
        }
        printf("\n");
    }
```

```
}
```

Footer

Footer navigation
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About
SY-BBA-CA-Sem-3-Practical-Slips/Practical Slip 20B.c at main · PritKalariya/SY-BBA-CA-Sem-3-Practical-Slips · GitHub

Silp 21A
Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 21A.c
@PritKalariya
PritKalariya C Practical Slips programs.
 1 contributor
174 lines (130 sloc) 3.08 KB

```
// Write a 'C' program to read an adjacency matrix of a directed graph and traverse using BFS


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

```
#define SIZE 40


struct queue {
  int items[SIZE];
  int front;
  int rear;
};


struct queue* createQueue();
void enqueue(struct queue* q, int);
int dequeue(struct queue* q);
void display(struct queue* q);
int isEmpty(struct queue* q);
void printQueue(struct queue* q);


struct node {
  int vertex;
  struct node* next;
};


struct node* createNode(int);


struct Graph {
  int numVertices;
  struct node** adjLists;
  int* visited;
};


void bfs(struct Graph* graph, int startVertex) {
  struct queue* q = createQueue();

  graph->visited[startVertex] = 1;
  enqueue(q, startVertex);

  while (!isEmpty(q)) {
    printQueue(q);
    int currentVertex = dequeue(q);
    printf("Visited %d\n", currentVertex);
    struct node* temp = graph->adjLists[currentVertex];

    while (temp) {
      int adjVertex = temp->vertex;

      if (graph->visited[adjVertex] == 0) {
        graph->visited[adjVertex] = 1;
        enqueue(q, adjVertex);
      }
      temp = temp->next;
    }
  }
}


struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}


struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));
  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }

  return graph;
}


void addEdge(struct Graph* graph, int src, int dest) {
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;


  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}


struct queue* createQueue() {
  struct queue* q = malloc(sizeof(struct queue));
  q->front = -1;
  q->rear = -1;
  return q;
}


int isEmpty(struct queue* q) {
  if (q->rear == -1)
    return 1;
  else
    return 0;
}


void enqueue(struct queue* q, int value) {
  if (q->rear == SIZE - 1)
    printf("\nQueue is Full!!");
  else {
    if (q->front == -1)
      q->front = 0;
    q->rear++;
    q->items[q->rear] = value;
  }
}


int dequeue(struct queue* q) {
  int item;
  if (isEmpty(q)) {
    printf("Queue is empty");
    item = -1;
  } else {
    item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
      printf("Resetting queue ");
      q->front = q->rear = -1;
    }
  }
  return item;
}


void printQueue(struct queue* q) {
  int i = q->front;

  if (isEmpty(q)) {
    printf("Queue is empty");
  } else {
    printf("\nQueue contains \n");
    for (i = q->front; i < q->rear + 1; i++) {
      printf("%d ", q->items[i]);
    }
  }
}

int main() {
  struct Graph* graph = createGraph(6);
  addEdge(graph, 0, 1);
  addEdge(graph, 0, 2);
  addEdge(graph, 1, 2);
  addEdge(graph, 1, 4);
  addEdge(graph, 1, 3);
  addEdge(graph, 2, 4);
  addEdge(graph, 3, 4);

  bfs(graph, 0);

  return 0;
}
```

Docs
Contact GitHub
Pricing
API
Training
Blog
About
SY-BBA-CA-Sem-3-Practical-Slips/Practical Slip 21A.c at main · PritKalariya/SY-BBA-CA-Sem-3-Practical-Slips · GitHub.
.

Slip 21B

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 21B.c
@PritKalariya
PritKalariya DS Practical Slip (21B).
1 contributor
32 lines (25 sloc) 760 Bytes

```
//Write a 'C' program Accept n elements from user store it in an array.
//Accept a value from the user and use linear/Sequential search method to check whether the value is present in array or not.
//Display proper message

#include<stdio.h>
#include<conio.h>

void main() {
	int arr[100], n, i, search;
```

```
    printf("Enter the number of elements you want in the array: ");
    scanf("%d", &n);

    for(i=0; i<n; i++) {
        printf("Enter value for index %d: ", i);
        scanf("%d", &arr[i]);
    }

    printf("\nEnter the number you want to search in the array: ");
    scanf("%d", &search);

    for(i=0; i<n; i++) {
        if(arr[i] == search) {
            printf("\n%d found at %d position.\n", search, i);
            break;
        }
    }

    if(i == n) {
        printf("\n%d not found in the array.\n");
    }
}
```

Footer

Footer navigation
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About
SY-BBA-CA-Sem-3-Practical-Slips/Practical Slip 21B.c at main · PritKalariya/SY-BBA-CA-Sem-3-Practical-Slips · GitHub

Silp 22A
Skip to content
Sign up
PritKalariya

/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 22A.c
@PritKalariya
PritKalariya C Practical Slips programs.
 1 contributor
63 lines (49 sloc) 827 Bytes

```
/*
Write a 'C' program which accept an Expression and check whether the expression is
Parenthesized or not using stack.
(Use Static/Dynamic implementation of Stack)
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int top = -1;
char stack[100];


void push(char);
void pop();
void find_top();


void main() {
        int i;
        char a[100];

        printf("Enter expression: ");
        scanf("%s", &a);

        for (i = 0; a[i] != '\0';i++) {
                if (a[i] == '(') {
```

```
            push(a[i]);
        }
        else if (a[i] == ')') {
            pop();
        }
    }
    find_top();
}


void push(char a)
{
    stack[top] = a;
    top++;
}


void pop() {
    if (top == -1) {
        printf("expression is invalid\n");
        exit(0);
    }
    else {
        top--;
    }
}


void find_top() {
    if (top == -1) {
        printf("\nexpression is valid\n");
    }
    else {
        printf("\nexpression is invalid\n");
    }
}
```

Contact GitHub
Pricing
API
Training
Blog
About
SY-BBA-CA-Sem-3-Practical-Slips/Practical Slip 22A.c at main · PritKalariya/SY-BBA-CA-Sem-3-Practical-Slips · GitHub

Silp 22B

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 22B.c
@PritKalariya
PritKalariya DS 22B
1 contributor
91 lines (68 sloc) 1.29 KB

```
//Write a 'C' program to count all non-zero elements, odd numbers and even numbers in the singly linked list.

#include<stdio.h>
#include<stdlib.h>

struct node{
	int data;
	struct node *link;
};

struct node *root = NULL;
```

```
void main() {
    int ch;

    while(1) {
        printf("\nMenu\n");
        printf("1. Insert.\n");
        printf("2. Count.\n");
        printf("3. Exit.\n");

        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch(ch) {
            case 1: insert();
                    break;

            case 2: count();
                    break;

            case 3: exit(1);

            default: printf("\nINVALID INPUT!!\n");
        }
    }
}

void insert() {
    struct node *temp;

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

    printf("Enter node data: ");
    scanf("%d", &temp->data);
    temp->link = NULL;

    if(root == NULL) {
        root = temp;
    }
    else {
        struct node* p;

        p = root;

        while(p->link != NULL) {
            p = p->link;
        }

        p->link = temp;
    }

    printf("\nData entered successfully.\n");
}

void count() {
    struct node *temp;
    int even, odd;

    even = odd = 0;

    temp = root;

    if(root == NULL) {
        printf("\nThe list is empty.\n");
    }
    else {
        while(temp->link != NULL) {
            if(temp->data % 2 == 0) {
                    even++;
            }
            else {
                    odd++;
            }
            temp = temp->link;
        }

        printf("\nEven numbers: %d", even);
        printf("\nOdd numbers: %d\n", odd);
    }
}
```

Footer

Footer navigation
Terms
Privacy
Security
Status
Docs
Contact GitHub

Pricing
API
Training
Blog
About
SY-BBA-CA-Sem-3-Practical-Slips/Practical Slip 22B.c at main · PritKalariya/SY-BBA-CA-Sem-3-Practical-Slips · GitHub

Silp 24B

Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 24B.c
@PritKalariya
PritKalariya DS 24B
 1 contributor
85 lines (65 sloc) 1.16 KB

```
// Write a 'C' program to remove last node of the singly linked list and insert it at the beginning of list.

#include<stdio.h>
#include<stdlib.h>

struct node {
	int data;
	struct node *link;
};

struct node *root = NULL;

void main() {
    insert(1);
    insert(2);
    insert(3);
    insert(4);
    insert(5);

    swap();

    display();
}

void insert(int data) {
    struct node *temp;

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

    temp->data = data;
    temp->link = NULL;

    if(root == NULL) {
        root = temp;
    }
    else {
        struct node *p;

        p = root;

        while(p->link != NULL) {
            p = p->link;
        }

        p->link = temp;
    }

    printf("\nData entered successfully.\n");
}

void swap() {
    struct node *l = root; // Last
    int temp;

    if(root == NULL || root->link == NULL) {
        printf("\nERROR!!\n");
    }
    else {
        while(l->link != NULL) {
            l = l->link;
        }

        temp = l->data;
        l->data = root->data;
        root->data = temp;

        printf("\nSwap successfull.\n");
    }
}

void display() {
    struct node *temp = root;

    if(temp == NULL) {
        printf("\nLIST IS EMPTY!!\n");
    }
    else {
        printf("\n");
        while(temp != NULL) {
            printf("%d\t", temp->data);
            temp = temp->link;
        }
        printf("\n");
    }
}
```

SY-BBA-CA-Sem-3-Practical-Slips/Practical Slip 24B.c at main · PritKalariya/SY-BBA-CA-Sem-3-Practical-Slips · GitHub

Silp 25B
Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 25B.c
@PritKalariya
PritKalariya DS 25B
 1 contributor
84 lines (64 sloc) 1.17 KB

```
/*Write a menu driven program using 'C' for singly linked list-
-	To create linked list.
-	To display linked list
*/

#include<stdio.h>
#include<stdlib.h>

struct node {
	int data;
	struct node* link;
};

struct node* root = NULL;

void main() {
	int ch;

	while(1) {
		printf("\nMenu\n");
		printf("1. Append\n");
		printf("2. Display\n");
```

```
        printf("3. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch(ch) {
            case 1: append();
                    break;

            case 2: display();
                    break;

            case 3: exit(1);

            default: printf("\nINVALID INPUT!!\n");
        }
    }
}

void append() {
    struct node *temp;

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

    printf("Enter node data: ");
    scanf("%d", &temp->data);
    temp->link = NULL;

    if(root == NULL) {
        root = temp;
    }
    else {
        struct node* p;

        p = root;

        while(p->link != NULL) {
            p = p->link;
        }

        p->link = temp;
    }

    printf("\nData entered successfully.\n");
```

```
}

void display() {
	struct node* temp;
	temp = root;

	if(temp == NULL) {
		printf("\nLIST IS EMPTY!!\n");
	}
	else {
		printf("\n");
		while(temp != NULL) {
			printf("%d\t", temp->data);
			temp = temp->link;
		}
		printf("\n");
	}
}
```

Footer

Footer navigation
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About

SY-BBA-CA-Sem-3-Practical-Slips/Practical Slip 25B.c at main · PritKalariya/SY-BBA-CA-Sem-3-Practical-Slips · GitHub

Silp 27 B
Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code

Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 27B.c
@PritKalariya
PritKalariya DS 27B
1 contributor
64 lines (49 sloc) 884 Bytes

```
//Write a 'C' program to create Doubly Link list and display it.

#include<stdio.h>
#include<conio.h>

struct node{
        struct node *left;
        int data;
        struct node *right;
};

struct node *root = NULL;

void main() {
        insert(1);
        insert(2);
        insert(3);
        insert(4);
        insert(5);

        display();
}

void insert(int data) {
        struct node *temp;

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

        temp->data = data;
        temp->left = temp->right = NULL;

        if(root == NULL) {
```

```
        root = temp;
    }
    else {
        struct node *p;

        p = root;

        while(p->right != NULL) {
            p = p->right;
        }

        p->right = temp;
        temp->left = p;
    }

    printf("\nData entered successfully.\n");
}

void display() {
    struct node *temp = root;

    if(temp == NULL) {
        printf("\nTHE LIST IS EMPTY!!\n");
    }
    else {
        while(temp != NULL) {
            printf("%d\t", temp->data);
            temp = temp->right;
        }
        printf("\n");
    }
}
```

Blog
About
SY-BBA-CA-Sem-3-Practical-Slips/Practical Slip 27B.c at main · PritKalariya/SY-BBA-CA-Sem-3-Practical-Slips · GitHub

Silp 28B
Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 28B.c
@PritKalariya
PritKalariya DS 28B
1 contributor
80 lines (62 sloc) 1.29 KB

```
//Write a 'C' program to read n integers and create two lists such that all positive numbers are in one list and negative numbers are in another list.
//Display both the lists.

#include<stdio.h>
#include<stdlib.h>

struct node {
	int data;
	struct node* link;
};

void main() {
	struct node *l1 = NULL, *l2 = NULL;
	int n, i, a[100];

	printf("\nEnter the number of nodes you want to enter: ");
	scanf("%d", &n);
```

```
    for(i=0; i<n; i++) {
        printf("Enter node data for node %d: ", i);
        scanf("%d", &a[i]);
    }

    for(i=0; i<n; i++) {
        if(a[i] > 0) {
            l1 = insert(l1, a[i]);
        }
        else {
            l2 = insert(l2, a[i]);
        }
    }

    printf("\nThe positive node list is: ");
    display(l1);

    printf("\nThe negative node list is: ");
    display(l2);
}

int insert(struct node *root ,int num) {
    struct node *temp;

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

    temp->data = num;
    temp->link = NULL;

    if(root == NULL) {
        root = temp;
    }
    else {
        struct node* p;

        p = root;

        while(p->link != NULL) {
            p = p->link;
        }

        p->link = temp;
    }
```

```
        return root;
}

void display(struct node *root) {
        struct node *temp = root;

        if(temp == NULL) {
                printf("\nLIST IS EMPTY!!\n");
        }
        else {
                printf("\n");
                while(temp != NULL) {
                        printf("%d\t", temp->data);
                        temp = temp->link;
                }
                printf("\n");
        }
}
```

Footer

Footer navigation
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About
SY-BBA-CA-Sem-3-Practical-Slips/Practical Slip 28B.c at main · PritKalariya/SY-BBA-CA-Sem-3-Practical-Slips · GitHub

Silp 29B
Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips

Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 29B.c
@PritKalariya
PritKalariya DS 29B
1 contributor
71 lines (52 sloc) 1.16 KB

```c
//Write a 'C' program to create Circular Singly Link list and display it.

#include<stdio.h>
#include<stdlib.h>

struct node{
        int data;
        struct node *link;
};

struct node *root = NULL;

void main() {
        int n;

        printf("Enter the number of node you want to enter: ");
        scanf("%d", &n);

        create(n);

        display();
}

void create(int n) {
    int i, data;
    struct node *p, *temp;

    if(n >= 1) {
        root = (struct node *)malloc(sizeof(struct node));
        printf("Enter data for node 1: ");
        scanf("%d", &data);

        root->data = data;
        root->link = NULL;

        p = root;

        for(i=2; i<=n; i++) {
            temp = (struct node *)malloc(sizeof(struct node));

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

            temp->data = data;
            temp->link = NULL;

            p->link = temp;
            p = temp;
        }

        p->link = root;
    }
}

void display() {
            struct node *temp;

            if(root == NULL) {
                        printf("\nList is empty.\n");
            }
            else {
                        temp = root;
                        printf("\n");
                        do{
                                    printf("%d\t", temp->data);
                                    temp = temp->link;
                        }while(temp != root);
                        printf("\n");
            }
}
```

Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About
SY-BBA-CA-Sem-3-Practical-Slips/Practical Slip 29B.c at main · PritKalariya/SY-BBA-CA-Sem-3-Practical-Slips · GitHub

Silp 30B
Skip to content
Sign up
PritKalariya
/
SY-BBA-CA-Sem-3-Practical-Slips
Public
Code
Issues
1
Pull requests
Actions
Projects
Security
Insights
SY-BBA-CA-Sem-3-Practical-Slips/Data Structure Practical Slips/Practical Slip 30B.c
@PritKalariya
PritKalariya DS 30B
 1 contributor
87 lines (68 sloc) 1.29 KB

```
// Write a 'C' program to sort elements of a singly linked list in ascending order and display the sorted List.

#include<stdio.h>
#include<stdlib.h>

struct node {
        int data;
        struct node* link;
```

```
};

struct node* root = NULL;

void main() {
        int n, i;
        printf("\nEnter the number of nodes: ");
        scanf("%d", &n);
        for(i=0; i<n; i++) {
                append();
        }

        printf("\nOrigginal linked list: \n");
        display();

        sort();

        printf("\nSorted linked list is: \n");
        display();
}

void append() {
        struct node *temp;

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

        printf("\nEnter node data: ");
        scanf("%d", &temp->data);
        temp->link = NULL;

        if(root == NULL) {
                root = temp;
        }
        else {
                struct node* p;

                p = root;

                while(p->link != NULL) {
                        p = p->link;
                }

                p->link = temp;
        }
        printf("\nData entered successfully.\n");
}

void display() {
        struct node* temp;
        temp = root;

        if(temp == NULL) {
                printf("\nLIST IS EMPTY!!\n");
        }
        else {
                printf("\n");
                while(temp != NULL) {
                        printf("%d\t", temp->data);
                        temp = temp->link;
                }
                printf("\n");
        }
}

void sort() {
        struct node *p, *q;
        int temp;

        for(p = root; p != NULL; p = p->link) {
                for(q = p->link; q != NULL; q = q->link) {
                        if(p->data > q->data) {
                                temp = p->data;
                                p->data = q->data;
                                q->data = temp;
                        }
                }
        }
}
```

Footer

Footer navigation

Terms

Privacy

Security

Status

Docs

Contact GitHub

Pricing
API
Training
Blog
About

SY-BBA-CA-Sem-3-Practical-Slips/Practical Slip 30B.c at main · PritKalariya/SY-BBA-CA-Sem-3-Practical-Slips · GitHub