Lab Assignment-13

ROLL: 2005535 | NAME: SAHIL SINGH | DATE: 15/11/21

QUES 1: [1] Write a menu driven program to create a one way inorder threaded binary tree and traverse the tree in inorder without using stack or recursion.

SOLUTION:

#include <stdio.h>

#include <stdlib.h>

typedef *struct* Node

{

*int* data;

*int* r\_thread;

*struct* Node \*right;

*struct* Node \*left;

} Node;

*void* insert(Node \*\*, *int*);

*void* inorder\_front(Node \*);

*int* main()

{

    Node \*root = NULL;

*int* choice, val;

    do

    {

        printf("1) Insert\n2) Inorder Display\n3) Exit\n->: ");

        scanf("%d", &choice);

        printf("\n");

        switch (choice)

        {

        case 1:

            printf("Enter value: ");

            scanf("%d", &val);

            insert(&root, val);

            break;

        case 2:

            inorder\_front(root);

            break;

        default:

            printf("Exiting...\n");

        }

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

    } while (choice >= 1 && choice <= 2);

    return 0;

}

*void* insert(Node \*\**root*, *int* *val*)

{

    Node \*temp = (Node \*)malloc(sizeof(Node));

    temp->data = *val*;

    temp->r\_thread = 0;

    temp->left = temp->right = NULL;

    if (!\**root*)

    {

        \**root* = temp;

        return;

    }

    Node \*ptrR = \**root*;

    Node \*ptr\_prev = NULL;

    Node \*threadNode = NULL;

    while (ptrR)

    {

        ptr\_prev = ptrR;

        if (ptrR->data >= *val*)

        {

            threadNode = ptrR;

            ptrR = ptrR->left;

        }

        else if (!ptrR->r\_thread)

            ptrR = ptrR->right;

        else

            ptrR = NULL;

    }

    temp->right = threadNode;

    if (threadNode)

        temp->r\_thread = 1;

    if (ptr\_prev->data > *val*)

        ptr\_prev->left = temp;

    else

    {

        ptr\_prev->right = temp;

        ptr\_prev->r\_thread = 0;

    }

}

*void* inorder\_front(Node \**root*)

{

*int* flag = 0;

    while (*root*)

    {

        while (*root*->left && !flag)

*root* = *root*->left;

        printf("%d->", *root*->data);

        if (*root*->right && *root*->r\_thread)

        {

            flag = 1;

*root* = *root*->right;

            continue;

        }

*root* = *root*->right;

        flag = 0;

    }

    printf("\b\b \n");

}

OUTPUT:

1) Insert

2) Inorder Display

3) Exit

->: 1

Enter value: 6

-------------------------------------------------

1) Insert

2) Inorder Display

3) Exit

->: 1

Enter value: 1

-------------------------------------------------

1) Insert

2) Inorder Display

3) Exit

->: 1

Enter value: 2

-------------------------------------------------

1) Insert

2) Inorder Display

3) Exit

->: 1

Enter value: 4

-------------------------------------------------

1) Insert

2) Inorder Display

3) Exit

->: 1

Enter value: 7

-------------------------------------------------

1) Insert

2) Inorder Display

3) Exit

->: 2

1->2->4->6->7 >

-------------------------------------------------

1) Insert

2) Inorder Display

3) Exit

->: 3

Exiting...

-------------------------------------------------

QUES 2: [2] Write a program to create an expression tree for a given postfix expression and traverse the tree to check the correctness.

SOLUTION:

#include <stdio.h>

#include <stdlib.h>

typedef *struct* Node

{

*char* data;

*struct* Node \*left;

*struct* Node \*right;

} Node;

typedef *struct* Node Node;

typedef *struct* Stack

{

    Node \*data;

*struct* Stack \*link;

} Stack;

*int* isEmpty\_stack(Stack \**stack*)

{

    if (!*stack*)

        return 1;

    return 0;

}

*void* push(Stack \*\**stack*, Node \**data*)

{

    Stack \*temp = (Stack \*)malloc(sizeof(Stack));

    temp->data = *data*;

    temp->link = \**stack*;

    \**stack* = temp;

}

Node \*pop(Stack \*\**stack*)

{

    if (isEmpty\_stack(\**stack*))

    {

        printf("\nUnderflow!");

        return NULL;

    }

    Stack \*temp = (\**stack*);

    \**stack* = (\**stack*)->link;

    Node \*val = temp->data;

    free(temp);

    return val;

    Node \*scanExpression(*char* \*);

*void* preorder(Node \*);

*void* inorder(Node \*);

}

Node \*scanExpression(*char* \**expression*)

{

    if (!*expression*)

        return NULL;

*int* i = 0;

*char* operations[6] = {'+', '-', '\*', '/', '^', '%'};

    Stack \*stack = NULL;

    while (*expression*[i] != '\0')

    {

        if ((*expression*[i] >= 'A' && *expression*[i] <= 'Z') ||

            (*expression*[i] >= 'a' && *expression*[i] <= 'z'))

        {

            Node \*temp = (Node \*)malloc(sizeof(Node));

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

            temp->data = *expression*[i];

            push(&stack, temp);

        }

        else

        {

            for (*int* j = 0; j < 6; j++)

            {

                if (operations[j] == *expression*[i])

                {

                    Node \*temp =

                        (Node \*)malloc(sizeof(Node));

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

                    temp->data = *expression*[i];

                    temp->right = pop(&stack);

                    temp->left = pop(&stack);

                    push(&stack, temp);

                    break;

                }

            }

        }

        i++;

    }

    return pop(&stack);

}

*void* preorder(Node \**root*)

{

    if (!*root*)

        return;

    printf("%c", *root*->data);

    preorder(*root*->left);

    preorder(*root*->right);

}

*void* inorder(Node \**root*)

{

    if (!*root*)

        return;

    inorder(*root*->left);

    printf("%c", *root*->data);

    inorder(*root*->right);

}

*int* main()

{

*char* \*input;

    printf("Enter expression: ");

    scanf(" %s", input);

    Node \*root = scanExpression(input);

    printf("\nInorder: ");

    inorder(root);

    printf("\nPreorder: ");

    preorder(root);

    printf("\n");

    return 0;

}

OUTPUT:

Enter expression: abcd^e-\*+

Inorder: a+b\*c^d-e

Preorder: +a\*b-^cde

QUES 3: [3] Write a program to create an expression tree for a given prefix expression and traverse the tree to check the correctness.

SOLUTION:

#include <stdio.h>

#include <stdlib.h>

typedef *struct* Node

{

*char* data;

*struct* Node \*left;

*struct* Node \*right;

} Node;

typedef *struct* Node Node;

typedef *struct* Stack

{

    Node \*data;

*struct* Stack \*link;

} Stack;

*int* isEmpty\_stack(Stack \**stack*)

{

    if (!*stack*)

        return 1;

    return 0;

}

*void* push(Stack \*\**stack*, Node \**data*)

{

    Stack \*temp = (Stack \*)malloc(sizeof(Stack));

    temp->data = *data*;

    temp->link = \**stack*;

    \**stack* = temp;

}

Node \*pop(Stack \*\**stack*)

{

    if (isEmpty\_stack(\**stack*))

    {

        printf("\nUnderflow!");

        return NULL;

    }

    Stack \*temp = (\**stack*);

    \**stack* = (\**stack*)->link;

    Node \*val = temp->data;

    free(temp);

    return val;

}

Node \*scanExpression(*char* \**expression*)

{

    if (!*expression*)

        return NULL;

*int* i = 0;

    for (; *expression*[i] != '\0'; i++)

        ;

*char* operations[6] = {'+', '-', '\*', '/', '^', '%'};

    Stack \*stack = NULL;

    while (i != -1)

    {

        if ((*expression*[i] >= 'A' && *expression*[i] <= 'Z') ||

            (*expression*[i] >= 'a' && *expression*[i] <= 'z'))

        {

            Node \*temp = (Node \*)malloc(sizeof(Node));

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

            temp->data = *expression*[i];

            push(&stack, temp);

        }

        else

        {

            for (*int* j = 0; j < 6; j++)

            {

                if (operations[j] == *expression*[i])

                {

                    Node \*temp =

                        (Node \*)malloc(sizeof(Node));

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

                    temp->data = *expression*[i];

                    temp->left = pop(&stack);

                    temp->right = pop(&stack);

                    push(&stack, temp);

                    break;

                }

            }

        }

        i--;

    }

    return pop(&stack);

}

*void* postorder(Node \**root*)

{

    if (!*root*)

        return;

    postorder(*root*->left);

    postorder(*root*->right);

    printf("%c", *root*->data);

}

*void* inorder(Node \**root*)

{

    if (!*root*)

        return;

    inorder(*root*->left);

    printf("%c", *root*->data);

    inorder(*root*->right);

}

*int* main()

{

*char* input[50];

    printf("Enter expression: ");

    scanf(" %s", input);

    Node \*root = scanExpression(input);

    printf("\nInorder: ");

    inorder(root);

    printf("\nPostorder: ");

    postorder(root);

    printf("\n");

    return 0;

}

OUTPUT:

Enter expression: +a\*b-^cde

Inorder: a+b\*c^d-e

Postorder: abcd^e-\*+

QUES 4: [4] Write a menu driven program to implement the following sorting algorithms:

1. Insertion Sort
2. Bubble Sort
3. Selection Sort
4. Merge Sort
5. Quick Sort

SOLUTION:

#include <stdio.h>

#include <time.h>

*void* merge(*int* \**arr*, *int* *left*, *int* *mid*, *int* *right*)

{

*int* aux[*right*];

*int* t1 = *left*, t2 = *mid* + 1;

*int* i = *left*;

    while (t1 <= *mid* && t2 <= *right*)

    {

        if (*arr*[t1] < *arr*[t2])

            aux[i++] = *arr*[t1++];

        else

            aux[i++] = *arr*[t2++];

    }

    while (t1 <= *mid*)

        aux[i++] = *arr*[t1++];

    while (t2 <= *right*)

        aux[i++] = *arr*[t2++];

    for (*int* k = *left*; k <= *right*; k++)

*arr*[k] = aux[k];

}

*void* swap(*int* \**a*, *int* \**b*)

{

*int* temp = \**a*;

    \**a* = \**b*;

    \**b* = temp;

}

*void* merge\_sort(*int* \**arr*, *int* *left*, *int* *right*)

{

    if (*left* >= *right*)

        return;

*int* mid = (*left* + *right*) / 2;

    merge\_sort(*arr*, *left*, mid);

    merge\_sort(*arr*, mid + 1, *right*);

    merge(*arr*, *left*, mid, *right*);

}

*void* selection\_sort(*int* \**arr*, *int* *len*)

{

*int* k;

    for (*int* i = 0; i < *len* - 1; i++)

    {

        k = i;

        for (*int* j = i + 1; j < *len*; j++)

        {

            if (*arr*[k] > *arr*[j])

                k = j;

        }

        swap(&*arr*[k], &*arr*[i]);

    }

}

*void* bubble\_sort(*int* \**arr*, *int* *len*)

{

*int* largest\_index;

    while (*len*--)

    {

        largest\_index = 0;

        for (*int* j = 1; j <= *len*; j++)

        {

            if (*arr*[largest\_index] < *arr*[j])

                largest\_index = j;

        }

        swap(&*arr*[largest\_index], &*arr*[*len*]);

    }

}

*void* insertion\_sort(*int* \**arr*, *int* *len*)

{

*int* key, j;

    for (*int* i = 1; i < *len*; i++)

    {

        key = *arr*[i];

        j = i;

        while (j > 0 && *arr*[j - 1] >= key)

        {

*arr*[j] = *arr*[j - 1];

            j--;

        }

*arr*[j] = key;

    }

}

*void* quick\_sort(*int* \**arr*, *int* *left*, *int* *right*)

{

    if (*left* >= *right*)

        return;

*int* small\_index = *left*;

    for (*int* i = *left* + 1; i <= *right*; i++)

    {

        if (*arr*[i] <= *arr*[*left*])

        {

            small\_index++;

            swap(&*arr*[i], &*arr*[small\_index]);

        }

    }

    swap(&*arr*[small\_index], &*arr*[*left*]);

    quick\_sort(*arr*, *left*, small\_index - 1);

    quick\_sort(*arr*, small\_index + 1, *right*);

}

*void* display(*int* \**arr*, *int* *len*)

{

    for (*int* i = 0; i < *len*; i++)

        printf("%d ", *arr*[i]);

    printf("\n");

}

*int* main()

{

    clock\_t t;

*double* time\_taken = ((*double*)t) / CLOCKS\_PER\_SEC;

*int* choice, len;

*int* arr[100];

    do

    {

        printf("1) Create a new list\n2) Insertion Sort\n3) Bubble Sort\n");

        printf("4) Selection Sort\n5) merge Sort\n6) Quick Sort\n7) Exit\n->: ");

        scanf("%d", &choice);

        printf("\n");

        switch (choice)

        {

        case 1:

            printf("Enter Size: ");

            scanf("%d", &len);

            printf("Enter values: ");

            for (*int* i = 0; i < len; i++)

                scanf("%d", &arr[i]);

            break;

        case 2:

            t = clock();

            insertion\_sort(arr, len);

            t = clock() - t;

            printf("Time taken (in seconds): %f\n", time\_taken);

            display(arr, len);

            break;

        case 3:

            t = clock();

            bubble\_sort(arr, len);

            t = clock() - t;

            printf("Time taken (in seconds): %f\n", time\_taken);

            display(arr, len);

            break;

        case 4:

            t = clock();

            selection\_sort(arr, len);

            t = clock() - t;

            printf("Time taken (in seconds): %f\n", time\_taken);

            display(arr, len);

            break;

        case 5:

            t = clock();

            merge\_sort(arr, 0, len - 1);

            t = clock() - t;

            printf("Time taken (in seconds): %f\n", time\_taken);

            display(arr, len);

            break;

        case 6:

            t = clock();

            quick\_sort(arr, 0, len - 1);

            t = clock() - t;

            printf("Time taken (in seconds): %f\n", time\_taken);

            display(arr, len);

            break;

        default:

            printf("Exiting...\n");

        }

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

    } while (choice >= 1 && choice <= 6);

    return 0;

}

OUTPUT:

1) Create a new list

2) Insertion Sort

3) Bubble Sort

4) Selection Sort

5) merge Sort

6) Quick Sort

7) Exit

->: 1

Enter *Size*: 10

Enter values: 10 88 9 34 234 56 78 33 95 28

-------------------------------------------

1) Create a new list

2) Insertion Sort

3) Bubble Sort

4) Selection Sort

5) merge Sort

6) Quick Sort

7) Exit

->: 2

Time taken (in *seconds*): 0.024000

9 10 28 33 34 56 78 88 95 234

-------------------------------------------

1) Create a new list

2) Insertion Sort

3) Bubble Sort

4) Selection Sort

5) merge Sort

6) Quick Sort

7) Exit

->: 3

Time taken (in *seconds*): 0.024000

9 10 28 33 34 56 78 88 95 234

-------------------------------------------

1) Create a new list

2) Insertion Sort

3) Bubble Sort

4) Selection Sort

5) merge Sort

6) Quick Sort

7) Exit

->: 4

Time taken (in *seconds*): 0.024000

9 10 28 33 34 56 78 88 95 234

-------------------------------------------

1) Create a new list

2) Insertion Sort

3) Bubble Sort

4) Selection Sort

5) merge Sort

6) Quick Sort

7) Exit

->: 5

Time taken (in *seconds*): 0.024000

9 10 28 33 34 56 78 88 95 234

-------------------------------------------

1) Create a new list

2) Insertion Sort

3) Bubble Sort

4) Selection Sort

5) merge Sort

6) Quick Sort

7) Exit

->: 6

Time taken (in *seconds*): 0.024000

9 10 28 33 34 56 78 88 95 234

-------------------------------------------

1) Create a new list

2) Insertion Sort

3) Bubble Sort

4) Selection Sort

5) merge Sort

6) Quick Sort

7) Exit

->: 7

Exiting...

-------------------------------------------