**LAPORAN PRAKTIKUM**

**SESI IX**

**PRAKTIKUM COMP6362 – DATA STRUCTURES**

**KELAS BC20**



Oleh :

2440059495 – Bismo Agung Tri Achmad Bramantyo

**SEMESTER GENAP 2020/2021  
BINA NUSANTARA UNIVERSITY**

**MALANG**

## **Kode Program**

|  |
| --- |
| Source Code |
| #include <stdio.h>  #include <string.h>  #include <stdlib.h>  struct data  {      int ID;      char name[20];      int loyalty;      struct data \*right\_child;      struct data \*left\_child;  };  struct data\* create\_node(char name[20], int year, int id)  {      struct data\* newnode;      newnode = malloc(sizeof(struct data));      newnode->ID = id;      newnode->loyalty = year;      strcpy(newnode->name, name);      newnode->left\_child = NULL;      newnode->right\_child = NULL;      printf("\n--Add New Data Success---\n\n");      getchar();      system("cls");      return newnode;  }  struct data\* insert\_node(struct data \*root, int key,char name[20], int loyalty)  {        if(root==NULL)          return create\_node(name, loyalty, key);      else if(key > root->ID){            root->right\_child = insert\_node(root->right\_child, key, name, loyalty);}      else {            root->left\_child = insert\_node(root->left\_child, key , name, loyalty);}        printf("\n--Add New Data Success---\n");      getchar();      system("cls");      return root;  }  struct data\* search(struct data \*root, int key)  {      if(root == NULL || root->ID == key)          return root;      else if(key > root->ID)          return search(root->right\_child, key);      else if (key < root->ID)          return search(root->left\_child, key);      else      printf("\n---ID not found---");    }  struct data\* find\_minimum(struct data \*root)  {      if(root == NULL)          return NULL;      else if(root->left\_child != NULL)          return find\_minimum(root->left\_child);      return root;  }  struct data\* delete\_node(struct data \*root, int key)  {        if(root==NULL)          printf("\nThere is no data\n");          return NULL;      if (key > root->ID)          root->right\_child = delete\_node(root->right\_child, key);      else if(key < root->ID)          root->left\_child = delete\_node(root->left\_child, key);      else      {            if(root->left\_child==NULL && root->right\_child==NULL)          {              printf("\n C%d %s is deleted\n", root->ID, root->name);              free(root);              return NULL;          }            else if(root->left\_child==NULL || root->right\_child==NULL)          {              struct node\* temp = root->right\_child;;              if(root->left\_child==NULL)                  temp = root->right\_child;              else                  temp = root->left\_child;                  printf("\n C%d %s is deleted\n", root->ID, root->name);              free(root);              return temp;          }                struct data \*temp = find\_minimum(root->right\_child);              root->ID = temp->ID;              root->right\_child = delete\_node(root->right\_child, temp->ID);      }      return root;  }  int height(struct data\* node)  {      if (node == NULL)          return 0;      else {            int lDepth = height(node->left\_child);          int rDepth = height(node->right\_child);              if (lDepth > rDepth)              return (lDepth + 1);          else              return (rDepth + 1);      }  }  void inorder(struct data \*root)  {      printf("\nIn-order: \n");      if(root!=NULL)      {          inorder(root->left\_child);          printf(" C%d %-10s [%d] \n", root->ID, root->name, root->loyalty);          inorder(root->right\_child);      }  }  void preorder(struct data \*root){      printf("\nPre-order: \n");      if(root != NULL){          printf(" C%d %-10s [%d] \n", root->ID, root->name, root->loyalty);          preorder(root->left\_child);          preorder(root->right\_child);      }  }  void postorder(struct data \*root){      printf("\nPost-order: \n");      if(root != NULL){          postorder(root->left\_child);          postorder(root->right\_child);          printf(" C%d %-10s [%d] \n", root->ID, root->name, root->loyalty);      }  }  int main(){      struct data\* r= NULL;      int parameter = 1;      int menu;      char name[20];      int year, id;      while(parameter == 1){          printf("Loyal Customer\n");          printf("###############\n\n");          printf("1. Add new data\n");          printf("2. Update a certain data\n");          printf("3. In-order, Pre-order and Post-order by Customer ID\n");          printf("4. Delete a certain data\n");          printf("5. Exit\n");          printf("\n>> Input Choice [1/2/3/4/5] : ");          scanf("%d", &menu);          switch(menu){                case 1:                if (height(r)>3){                  printf("\nMaximum Tree Height is 3\n");              }              else{                do{              printf("Input customer ID C[1-9][0-9]: C");              scanf("%d", &id); getchar();              }while(id < 10 || id > 99);              do{              printf("\nInput customer name: ");              scanf("%[^\n]", &\*name); getchar();              }while(strlen(name)>20 || strlen(name)<3 );                do{              printf("\ntotal years of loyality [0..100]: ");              scanf("%d", &year); getchar();              }while(year<0 || year>100);              insert\_node(r, id, name, year);                }                break;              case 2:                    printf("\nInput customer id: C");                  scanf("%d", &id);                  search(r, id);                  getchar();                  system("cls");                break;              case 3:              if(r != NULL){              preorder(r);              inorder(r);              postorder(r);              }              else{                  getchar();                  printf("\nThere is no data\n");                  getchar();                  system("cls");              }              break;              case 4:              if(r == NULL){                  printf("\nThere is no data\n");                  getchar();              system("cls");              getchar();              }              else{                printf("Input customer ID C[1-9][0-9]: C");              scanf("%d", &id);              delete\_node(r,id);              getchar();              system("cls");              getchar();              }              break;              case 5:              parameter = 0;              printf("\nProgramm Closed, Thank you\n");              break;          }      }  } |

## **Deskripsi Program**

1. Library

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

1. Struct

struct data

{

    int ID;

    char name[20];

    int loyalty;

    struct data \*right\_child;

    struct data \*left\_child;

};

Ada 3 variabel utama yaitu ID untuk nomor identitas , name untuk nama customer, dan loyalty untuk tahun langganan. Ada 2 pointer utama right dan left untuk pointer child pada tree.

1. Fungsi
2. Membuat node baru

struct data\* create\_node(char name[20], int year, int id)

{

    struct data\* newnode;

    newnode = malloc(sizeof(struct data));

    newnode->ID = id;

    newnode->loyalty = year;

    strcpy(newnode->name, name);

    newnode->left\_child = NULL;

    newnode->right\_child = NULL;

    printf("\n--Add New Data Success---\n\n");

    getchar();

    system("cls");

    return newnode;

}

Ada 3 passing parameter , nama , tahun dan id nya. Dilakukan pendefinisian node, lalu subtitusi data ke node dan yang terakhir peng-NULL an pointer.

1. Insert node

struct data\* insert\_node(struct data \*root, int key,char name[20], int loyalty)

{

    if(root==NULL)

        return create\_node(name, loyalty, key);

    else if(key > root->ID){

        root->right\_child = insert\_node(root->right\_child, key, name, loyalty);}

    else {

        root->left\_child = insert\_node(root->left\_child, key , name, loyalty);}

    printf("\n--Add New Data Success---\n");

    getchar();

    system("cls");

    return root;

}

Untuk insert sendiri berfungsi pada peletakan node baru, apabila belum ada node, maka akan dilakuakn create node, tetapi apabila sudah ada maka akan dilakukan insert sesuai dengan ketentutan BST, apabila lebih kecil maka akan berada di left child begitu juga sebaliknya.

1. Search node

struct data\* search(struct data \*root, int key)

{

    if(root == NULL || root->ID == key)

        return root;

    else if(key > root->ID)

        return search(root->right\_child, key);

    else if (key < root->ID)

        return search(root->left\_child, key);

    else

    printf("\n---ID not found---");

untuk search sendiri berfungsi untuk mencari node, ketika root kosong atau id pada root sama dengan yang dicari maka posisi tersebut akan terletak di root, namun apabila tidak, maka akan dilihat, jika data lebih kecil maka cari ke left child, jika tidak maka cari ke right child.

1. Mencari data terkecil

struct data\* find\_minimum(struct data \*root)

{

    if(root == NULL)

        return NULL;

    else if(root->left\_child != NULL)

        return find\_minimum(root->left\_child);

    return root;

}

Fungsi find minimum berfungsi untuk mencari nilai terkecil dari data yang ada, digunakan pada saat deleting node. Ketika root null, maka nilai null akan dikembalikan jika tidak maka akan dicari di anak sisi kiri, lalu pointer dikembalikan seperti semula.

1. Deleteing node

struct data\* delete\_node(struct data \*root, int key)

{

    if(root==NULL)

        printf("\nThere is no data\n");

        return NULL;

    if (key > root->ID)

        root->right\_child = delete\_node(root->right\_child, key);

    else if(key < root->ID)

        root->left\_child = delete\_node(root->left\_child, key);

    else

    {

        if(root->left\_child==NULL && root->right\_child==NULL)

        {

            printf("\n C%d %s is deleted\n", root->ID, root->name);

            free(root);

            return NULL;

        }

        else if(root->left\_child==NULL || root->right\_child==NULL)

        {

            struct node\* temp = root->right\_child;;

            if(root->left\_child==NULL)

                temp = root->right\_child;

            else

                temp = root->left\_child;

                printf("\n C%d %s is deleted\n", root->ID, root->name);

            free(root);

            return temp;

        }

            struct data \*temp = find\_minimum(root->right\_child);

            root->ID = temp->ID;

            root->right\_child = delete\_node(root->right\_child, temp->ID);

    }

    return root;

}

1. Mencari height

int height(struct data\* node)

{

    if (node == NULL)

        return 0;

    else {

        int lDepth = height(node->left\_child);

        int rDepth = height(node->right\_child);

        if (lDepth > rDepth)

            return (lDepth + 1);

        else

            return (rDepth + 1);

    }

}

1. Transversal

void inorder(struct data \*root)

{

    printf("\nIn-order: \n");

    if(root!=NULL)

    {

        inorder(root->left\_child);

        printf(" C%d %-10s [%d] \n", root->ID, root->name, root->loyalty);

        inorder(root->right\_child);

    }

}

void preorder(struct data \*root){

    printf("\nPre-order: \n");

    if(root != NULL){

        printf(" C%d %-10s [%d] \n", root->ID, root->name, root->loyalty);

        preorder(root->left\_child);

        preorder(root->right\_child);

    }

}

void postorder(struct data \*root){

    printf("\nPost-order: \n");

    if(root != NULL){

        postorder(root->left\_child);

        postorder(root->right\_child);

        printf(" C%d %-10s [%d] \n", root->ID, root->name, root->loyalty);

    }

}

1. Main

int main(){

    struct data\* r= NULL;

    int parameter = 1;

    int menu;

    char name[20];

    int year, id;

    while(parameter == 1){

        printf("Loyal Customer\n");

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

        printf("1. Add new data\n");

        printf("2. Update a certain data\n");

        printf("3. In-order, Pre-order and Post-order by Customer ID\n");

        printf("4. Delete a certain data\n");

        printf("5. Exit\n");

        printf("\n>> Input Choice [1/2/3/4/5] : ");

        scanf("%d", &menu);

        switch(menu){

            case 1:

            if (height(r)>3){

                printf("\nMaximum Tree Height is 3\n");

            }

            else{

            do{

            printf("Input customer ID C[1-9][0-9]: C");

            scanf("%d", &id); getchar();

            }while(id < 10 || id > 99);

            do{

            printf("\nInput customer name: ");

            scanf("%[^\n]", &\*name); getchar();

            }while(strlen(name)>20 || strlen(name)<3 );

            do{

            printf("\ntotal years of loyality [0..100]: ");

            scanf("%d", &year); getchar();

            }while(year<0 || year>100);

            insert\_node(r, id, name, year);

            }

            break;

            case 2:

                printf("\nInput customer id: C");

                scanf("%d", &id);

                search(r, id);

                getchar();

                system("cls");

            break;

            case 3:

            if(r != NULL){

            preorder(r);

            inorder(r);

            postorder(r);

            }

            else{

                getchar();

                printf("\nThere is no data\n");

                getchar();

                system("cls");

            }

            break;

            case 4:

            if(r == NULL){

                printf("\nThere is no data\n");

                getchar();

            system("cls");

            getchar();

            }

            else{

            printf("Input customer ID C[1-9][0-9]: C");

            scanf("%d", &id);

            delete\_node(r,id);

            getchar();

            system("cls");

            getchar();

            }

            break;

            case 5:

            parameter = 0;

            printf("\nProgramm Closed, Thank you\n");

            break;

        }

    }

}

## **Bukti Presentasi**