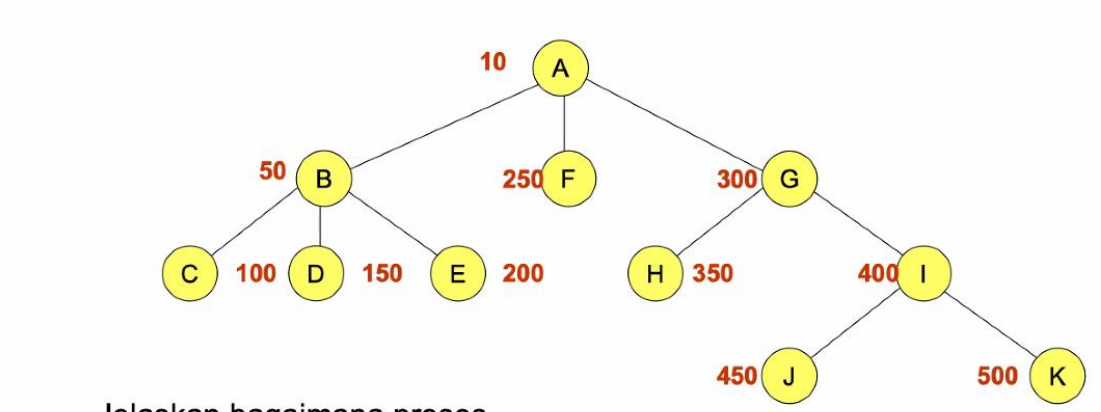
LATIHAN IMPLEMENTASI TREE

KELOMPOK 4:

- DANGIANG RAKEAN A (2106902)
- KHAIRUNNISA (2103683)
- SAJIDA I'TIKAFAH L (2102116)
- TIA AULIA MARHAMAH (2100301)
- WENDI KARDIAN (2100016)

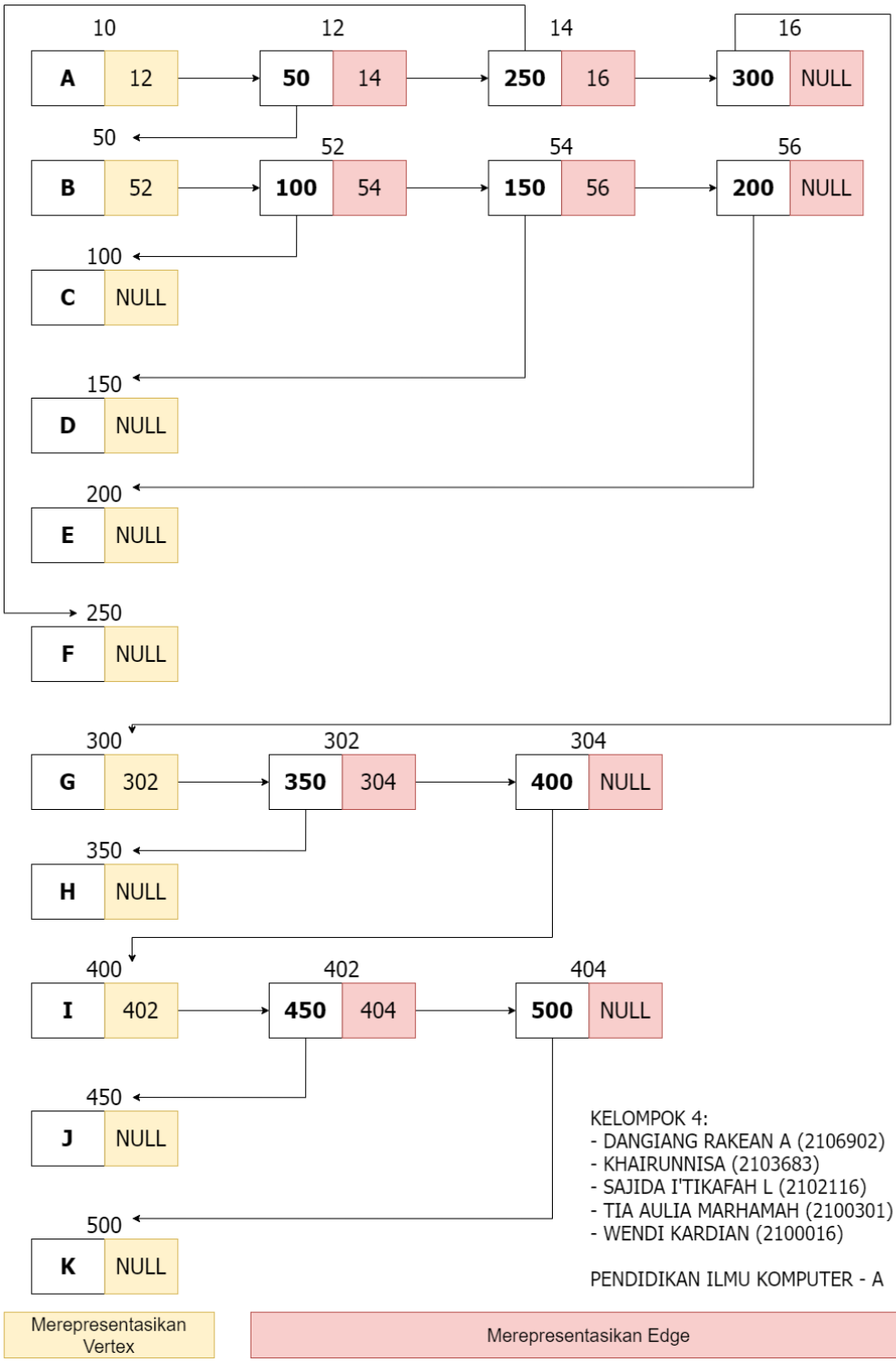
PENDIDIKAN ILMU KOMPUTER – A



Jelaskan bagaimana proses implementasi tree berikut!

a. Implementasi 1
b. Implementasi 2

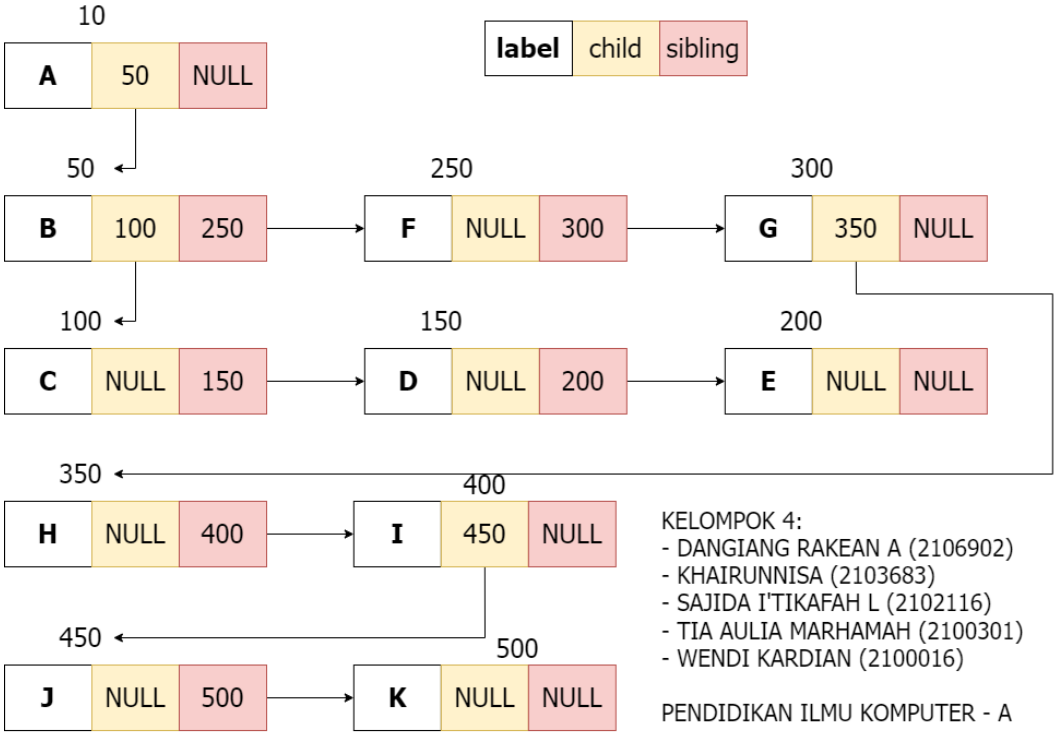
IMPLEMENTASI 1



KELOMPOK 4:
- DANGIANG RAKEAN A (2106902)
- KHAIRUNNISA (2103683)
- SAJIDA I'TIKAFAH L (2102116)
- TIA AULIA MARHAMAH (2100301)
- WENDI KARDIAN (2100016)

PENDIDIKAN ILMU KOMPUTER - A

IMPLEMENTASI 2



KELOMPOK 4:
- DANGIANG RAKEAN A (2106902)
- KHAIRUNNISA (2103683)
- SAJIDA I'TIKAFAH L (2102116)
- TIA AULIA MARHAMAH (2100301)
- WENDI KARDIAN (2100016)

PENDIDIKAN ILMU KOMPUTER - A

Source Code Implementasi 1

Untuk lebih jelas dapat kunjungi link Github : [Data-Structure-using-C/implementasi1.c at master · wendikardian/Data-Structure-using-C (github.com)]

```c
// STRUKTUR DATA TREE STATIC
// WENDI KARDIAN (2100016) - Pendidikan Ilmu Komputer - A

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

/*struct untuk vertex*/
typedef struct simpul{
    struct ruas *jalur;
    struct simpul *nextVertex;
    char label;
    char id[2];
}simpul;

/*struct untuk edge*/
typedef struct ruas{
    struct ruas *nextEdge;
    struct simpul *vertexTujuan;
    char bobot[2];
}ruas;

simpul *awal = NULL;

// Membuat vertex baru
simpul *createVertex (char a, char b[]){
    simpul *simpulBaru = (simpul*)malloc(sizeof(simpul));
    simpulBaru->label = a;
    strcpy(simpulBaru->id,b);
    simpulBaru->jalur = NULL;
    simpulBaru->nextVertex = NULL;

    return simpulBaru;
}

/*fungsi untuk menemukan vertex didalam graph*/
simpul *cariSimpul(char a) {
    simpul *bantu = awal;
```

```
    if(bantu != NULL){
        while(bantu->nextVertex != NULL){
            if(bantu->label == a){
                break;
            }

            bantu = bantu->nextVertex;
        }
    }

    return bantu;
}

void tambahVertex(char a, char b[]){ //menanbahkan vertex
berdasarkan label dan idnya lalu disambungkan
    simpul *prev = cariSimpul(a);

    if(prev == NULL){
        simpul *baru = createVertex(a, b);
        awal = baru;
    }
    else{
        if((prev->nextVertex == NULL) && (prev->label != a)){
            simpul *baru = createVertex(a, b);
            prev->nextVertex = baru;
        }
    }
}

void createEdge (simpul *a, simpul *t, char bobot[]){ //membuat
hubungan antar vertex
    ruas *newEdge = (ruas*)malloc(sizeof(ruas));
    strcpy(newEdge->bobot,bobot);
    newEdge->nextEdge = NULL;
    newEdge->vertexTujuan = t;

    if (a->jalur == NULL){
        a->jalur = newEdge;
    }
    else{
        ruas *jalurAkhir = a->jalur;
        while (jalurAkhir->nextEdge != NULL){
            jalurAkhir = jalurAkhir->nextEdge;
        }
        jalurAkhir->nextEdge = newEdge;
    }
```

```
}

void tambahEdge(char Vasal, char nilaiEdge[], char Vtujuan){
//menambahkan edge
    simpul *a,*t;
    a = cariSimpul(Vasal);
    t = cariSimpul(Vtujuan);
    createEdge(a,t,nilaiEdge);
}

// Display the graph
void display(){
    simpul *tempSimpul = awal;
    printf("|-------------------------------------------------------
----| \n");
    printf("|                 NILAI
TREE                              | \n");
    printf("|-------------------------------------------------------
----| \n");
    if (tempSimpul != NULL) {
        while (tempSimpul != NULL){
            if(tempSimpul->jalur != NULL){
                printf("\n Vertex %c yang memiliki id %s Memiliki
Child : \n", tempSimpul->label, tempSimpul->id);
                ruas *tempEdge = tempSimpul->jalur;
                while (tempEdge != NULL){
                    printf(" ------ vertex %c \n",tempEdge-
>vertexTujuan->label);
                    tempEdge = tempEdge->nextEdge;
                }
                printf("-----------------------------------------
---------------\n");
                printf("\n");
                tempSimpul = tempSimpul->nextVertex;
            }else{
                tempSimpul = tempSimpul->nextVertex;
            }
        }
    }
    else{
        printf("Graph Kosong");
    }
}

void checkVertex(){
    simpul *tempSimpul = awal;
```

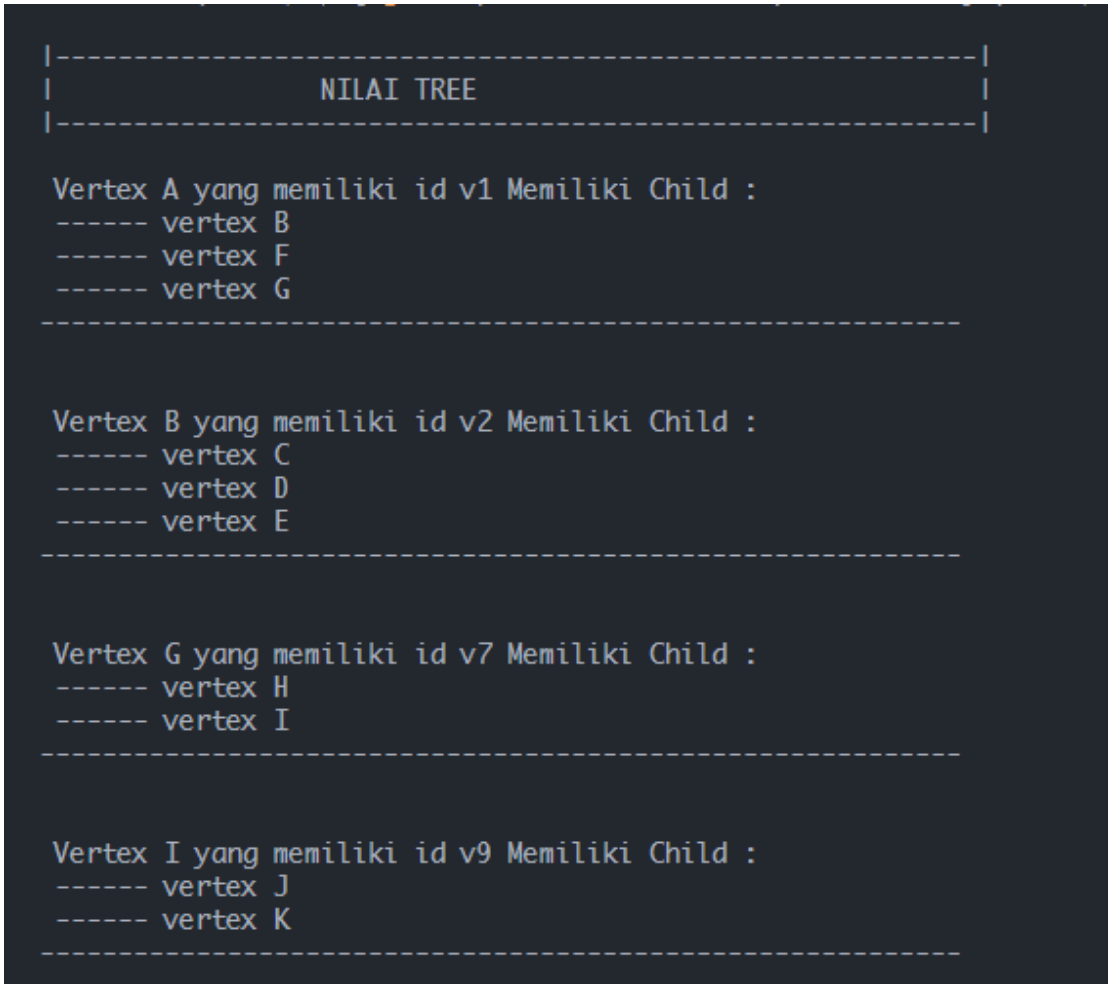
```
    while(tempSimpul != NULL){
        printf("%c\n", tempSimpul->label);
        tempSimpul = tempSimpul->nextVertex;
    }
}

int main(){

    tambahVertex('A', "v1");
    tambahVertex('B', "v2");
    tambahVertex('C', "v3");
    tambahVertex('D', "v4");
    tambahVertex('E', "v5");
    tambahVertex('E', "v5");
    tambahVertex('F', "v6");
    tambahVertex('G', "v7");
    tambahVertex('H', "v8");
    tambahVertex('I', "v9");
    tambahVertex('J', "v10");
    tambahVertex('K', "v11");
    tambahEdge('A',"5",'B');
    tambahEdge('A',"e2",'F');
    tambahEdge('A',"e2",'G');
    tambahEdge('B',"e2",'C');
    tambahEdge('B',"e3",'D');
    tambahEdge('B',"e4",'E');
    tambahEdge('G',"4",'H');
    tambahEdge('G',"7",'I');
    tambahEdge('I',"3",'J');
    tambahEdge('I',"9",'K');
    display();

    return 0;
}
```

Output SC Implementasi 1 :



Source Code Implementasi 2

Untuk lebih jelas dapat kunjungi link Github : [Data-Structure-using-C/implementasi2.c at master · wendikardian/Data-Structure-using-C (github.com)]

```c
// TREE using C
// IMPLEMENTASI Ke -2


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

typedef struct vertex{
    char label;
    struct vertex *sibling;
    struct vertex *child;
}Vertex;


Vertex *root = NULL;

Vertex * createVertex(char label){
    Vertex *new = (Vertex*)malloc(sizeof(Vertex));
    new->label =  label;
    new->sibling = NULL;
    new->child = NULL;
    return new;
}


void addChild(char c, Vertex *root){
    if(root != NULL){
        Vertex *new = createVertex(c);
        if(root->child == NULL){
            new->sibling = NULL;
            root->child = new;
        }else{
            if(root->child->sibling == NULL){
                new->sibling = root->child;
                root->child->sibling = new;
            }else{
                Vertex *last = root->child;
                while(last->sibling != root->child){
                    last = last->sibling;
                }
```

```
                new->sibling = root->child;
                last->sibling = new;
            }
        }
    }
}

Vertex *findVertex(char c, Vertex *root){
    Vertex *result = NULL;
    if(root != NULL){
        if(root->label == c){
            result = root;
        }else{
            Vertex *ptr = root->child;
            if(ptr != NULL){
                if(ptr->sibling == NULL){
                    if(ptr->label == c){
                        result =  ptr;
                    }else{
                        result = findVertex(c, ptr);
                    }
                }else{
                    int find = 0;
                    while(ptr->sibling != root->child && (find ==
0)){
                        if(ptr->label == c){
                            result = ptr;
                            find = 1;
                        }else{
                            result = findVertex(c, ptr);
                            ptr = ptr->sibling;
                        }
                    }
                if(find == 0){
                    if(ptr->label == c){
                        result = ptr;
                    }else{
                        result = findVertex(c, ptr);
                    }
                }
                }
            }
        }
    }
    return result;
}
```

```
void dellAll(Vertex *root){
    if(root != NULL){
        if(root->child != NULL){
            if(root->child->sibling == NULL){
                dellAll(root->child);
                free(root);
            }else{
                Vertex *ptr;
                Vertex *help;
                ptr = root->child;
                while(ptr->sibling != root->child){
                    help = ptr;
                    ptr = ptr->sibling;
                    dellAll(help);
                }
            }
            free(root);
        }else{
            free(root);
        }
    }
}


void dellChild(char c, Vertex *root){
    if(root != NULL) {
        Vertex *delete = root->child;
        if(delete != NULL){
            if(delete->sibling == NULL){
                if(root->child->label == c){
                    dellAll(root->child);
                    root->child == NULL;
                }else{
                    printf("Vertex child not found\n");
                }
            }else{
                Vertex *ptr = NULL;
                int find = 0;
                while((delete->sibling != root->child) && (find ==
0)){
                    if(delete->label == c){
                        find = 1;
                    }else{
                        ptr = delete;
                        delete = delete->sibling;
```

```
                    }
                }

                if(find == 0 && (delete->label == c)){
                    find = 1;
                }

                if(find == 1){
                    Vertex *last = root->child;
                    while(last->sibling != root->child){
                        last = last->sibling;
                    }

                    if(ptr == NULL){
                        if((delete->sibling == last) && (last-
>sibling == root->child)){
                            root->child = last;
                            last->sibling = NULL;
                        }else{
                            root->child = delete->sibling;
                            last->sibling = root->child;
                        }
                    }else{
                        if((ptr == root->child) && (last->sibling ==
root->child )){
                            root->child->sibling = NULL;
                        }else{
                            ptr->sibling = delete->sibling;
                            delete->sibling = NULL;
                        }
                    }
                    dellAll(delete);
                }else{
                    printf("Vertex child doesnt exist\n");
                }
            }
        }
    }
}

void preOrder(Vertex *root){
    if(root != NULL){
        printf(" %c", root->label);
        Vertex *ptr = root->child;
        if(ptr != NULL){
            if(ptr->sibling == NULL){
```

```
                preOrder(ptr);
            }else{
                while(ptr->sibling != root->child){
                    preOrder(ptr);
                    ptr = ptr->sibling;
                }
                preOrder(ptr);
            }
        }
    }
}

void postOrder(Vertex *root){
    if(root != NULL){
        Vertex *ptr = root->child;
        if(ptr != NULL){
            if(ptr->sibling == NULL){
                postOrder(ptr);
            }else{
                while(ptr->sibling != root->child){
                    postOrder(ptr);
                    ptr = ptr->sibling;
                }
                postOrder(ptr);
            }
        }
        printf(" %c", root->label);
    }
}


int main(){
    root = createVertex('A');
    addChild('B', root);
    addChild('F', root);
    addChild('G', root);
    Vertex *rootB = findVertex('B', root);
    Vertex *rootG = findVertex('G', root);
    addChild('C', rootB);
    addChild('D', rootB);
    addChild('E', rootB);
    addChild('H', rootG);
    addChild('I', rootG);
    Vertex *rootI = findVertex('I', rootG);
    addChild('J', rootI);
    addChild('K', rootI);
```

```
    printf("============================\n        TREEE
\n============================\n\n");
    printf("PRE ORDER : \n");
    preOrder(root);
    printf("\nPOST ORDER : \n");
    postOrder(root);
    return 0;
}
```

Output Dari Source Code Implementasi 2 :

```
PS D:\UPI\SEMESTER 2\STRUKTUR DATA\Pertemuan 12> cd "d:\UPI\SEMESTER 2\STRUKTUR DATA\Perte
muan 12\" ; if ($?) { gcc tempCodeRunnerFile.c -o tempCodeRunnerFile } ; if ($?) { .\tempC
odeRunnerFile }
============================
        TREEE
============================

PRE ORDER :
 A B C D E F G H I J K
POST ORDER :
 C D E B F H J K I G A
```