Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра системного проектування сервісів

ЗВІТ

про виконання комп'ютерного практикуму № 1
з дисципліни «Алгоритми і структури даних»

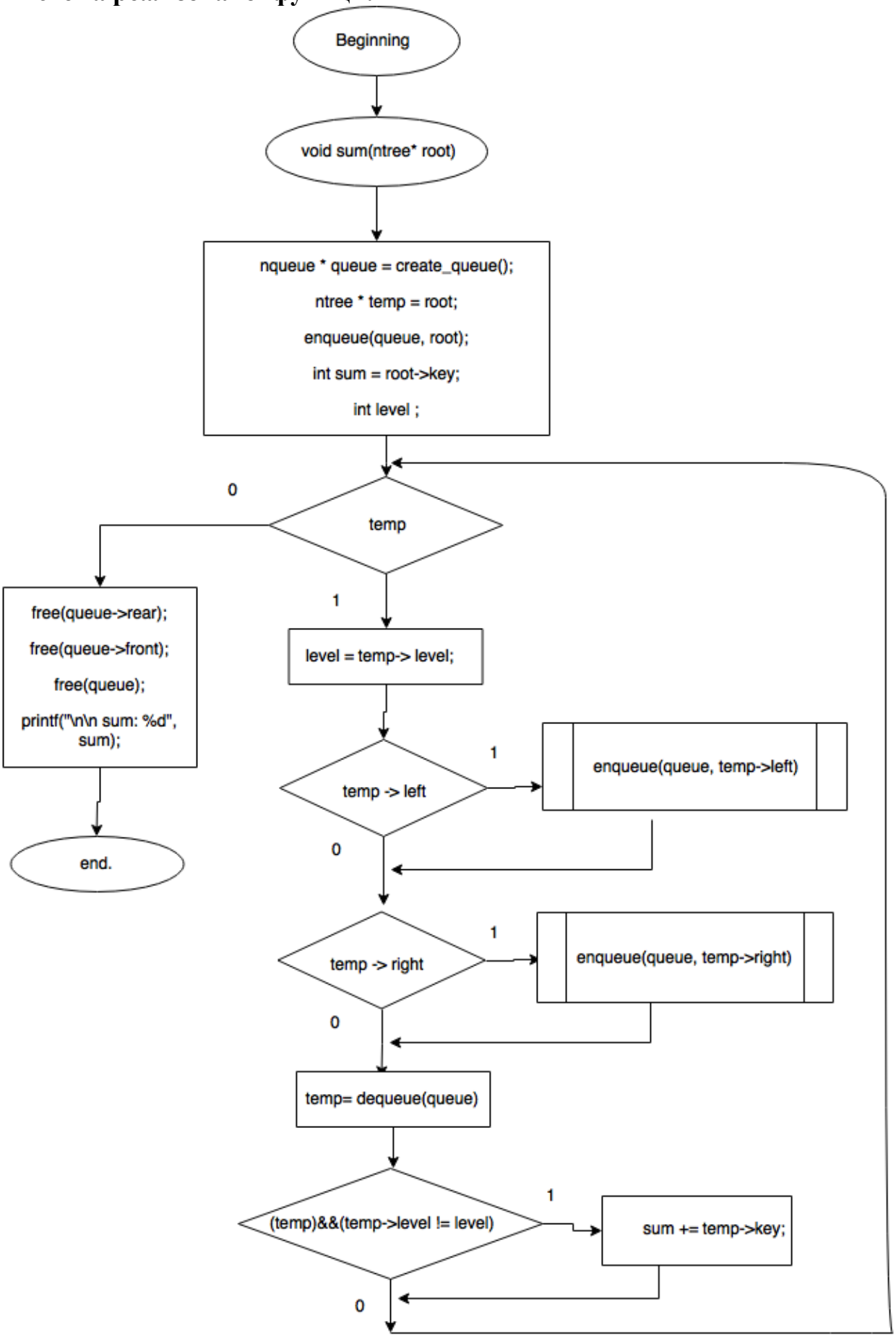
Виконав:
Студент І курсу
Групи ДА-72
Хоменко Є.С
Варіант № 23

Перевірив:
Караюз І.В

Київ – 2018

Завдання: Обчислити суму ключів лівих вузлів дерева, лівий вузол – це перший вузол рівня дерева.

Блок – схема реалізованої функції:



Лістинг програми:

// Main.c

```c
#include "MainFile.h"
#include <stdio.h>
#include <stdlib.h>

#define N 20

int main(){

    FILE *fp = fopen("/Users/ievgenkhonenko/Desktop/MyTxt.txt","r+");
    int *c = (int*) malloc(N*sizeof(int));

    for(int i = 0;i < N; i++){
        fscanf(fp,"%d",&c[i]);
    }
    fclose(fp);

    ntree * root = NULL;

    for (int i=0; i<N; i++) {
        int key = c[i];
        printf("%d\t", key);
        root = build_tree(root, root, key);
    }

    printf("\n");
    printf("inorder: \n");
    in_order(root);
    printf("\n");
    printf("preorder:\n");
    pre_order(root);
    printf("\n");
    printf("postorder:\n");
    post_order(root);
    printf("\n");
    printf("levelorder:\n");
    level_order(root);
    sum(root);
    printf("\n\n%d nodes were deleted\n", free_tree(root));
    free(c);
    return 0;
}
```

// MainFile.h

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

typedef struct node_tree{
    int key;
    int level;
    struct node_tree * left, * right, * next;
} ntree;

typedef struct queue{
    ntree * rear,* front;
}nqueue;
```

```
ntree * build_tree(ntree *, ntree *, int);
void in_order(ntree *);
void pre_order(ntree *);
void post_order(ntree *);
int free_tree(ntree *);
ntree* create_node(ntree * );
void enqueue (nqueue*, ntree *);
ntree* dequeue (nqueue*);
void level_order(ntree*);
void sum(ntree*);

ntree * build_tree(ntree * root, ntree * node, int key){
    if (!node){

        node = (ntree *) malloc(sizeof(ntree));
        if (!node) {
            printf("\nERROR: Bad allocation!");
            exit(1);
        }

        node->left = node->right = NULL;
        node->key = key;

        if (!root){
            node -> level = 0;
            return node; // create root
        }
        node-> level = root->level + 1;
        if (key < root->key) root->left = node;
        else root->right = node;
        return node;
    }
    if (key < node->key) build_tree(node, node->left, key);
    else build_tree(node, node->right, key);

    return root;
}


void in_order(ntree * h){
    if (h->left) in_order(h->left);
    printf("%d\t", h->key);
    if (h->right) in_order(h->right);
}
void pre_order(ntree * h){
    printf("%d\t",h->key);
    if(h->left) pre_order(h->left);
    if(h->right) pre_order(h->right);
}
void post_order(ntree * h){
    if(h->left) post_order(h->left);
    if(h->right) post_order(h->right);
    printf("%d\t",h->key);
}

int free_tree(ntree * h){
    static int c = 0;

    if (h->left) free_tree(h->left);
    if (h->right) free_tree(h->right);

    if (h) {
```

```
      c++;

     free(h);
    }
    return c;
}

ntree * create_node(ntree * node){
    ntree * new_node = node;
    new_node -> next = NULL;
    return new_node;
}

nqueue* create_queue(){
    nqueue * queue = (nqueue*)malloc(sizeof(nqueue));
    queue -> rear = NULL;
    queue -> front = NULL;
    return queue;
}

void enqueue ( nqueue* queue, ntree * node ){
    ntree * temp = create_node(node);
    if (!(queue->rear))
    {
        queue->front = queue->rear = temp;
        return;
    }
    (queue->rear)->next = temp;
    queue->rear = temp;
}
ntree * dequeue(nqueue * queue){
    if(!(queue -> front))  return NULL;
    ntree * temp = queue -> front;
    queue -> front = (queue -> front) -> next;
    if (!(queue->front))  queue->rear = NULL;
    return temp;
}

void level_order(ntree* root){
    nqueue * queue = create_queue();
    ntree * temp = root;
    enqueue(queue, root);
    int i = 0;
    while(temp){
         if(i > 0) printf("%d(%d) ", temp->key, temp -> level);
        if(temp -> left){
            enqueue(queue, temp->left);
        }
        if (temp->right) {
            enqueue(queue, temp->right);
        }
         temp =  dequeue(queue);
         i++;
    }
    free(queue->rear);
    free(queue->front);
    free(queue);
    }

void sum(ntree* root){
    nqueue * queue = create_queue();
    ntree * temp = root;
    enqueue(queue, root);
```

```
    int level ;
    while(temp){

       level = temp-> level;
        if(temp -> left){
            enqueue(queue, temp->left);
        }
        if (temp->right) {
            enqueue(queue, temp->right);
        }
        temp =  dequeue(queue);
        if((temp)&&(temp->level != level)){
            sum += temp->key;
        }
    }
    free(queue->rear);
    free(queue->front);
    free(queue);
    printf("\n\n sum: %d", sum);

}
```

Результати роботи програми:

```
69  415 241 403 666 56  482 746 932 756 179 185 790
839 978 25  359 522 831 101
inorder:
25  56  69  101 179 185 241 359 403 415 482 522 666
746 756 790 831 839 932 978
preorder:
69  56  25  415 241 179 101 185 403 359 666 482 522
746 932 756 790 839 831 978
postorder:
25  56  101 185 179 359 403 241 522 482 831 839 790
756 978 932 746 666 415 69
levelorder:
69(0) 56(1) 415(1) 25(2) 241(2) 666(2) 179(3) 403(3)
482(3) 746(3) 101(4) 185(4) 359(4) 522(4) 932(4)
756(5) 978(5) 790(6) 839(7) 831(8)

 sum: 3646

20 nodes were deleted
Program ended with exit code: 0
```

Висновки: У ході виконання комп’ютерного практикуму мовою С був написаний код, що ініціалізує динамічну нелінійну структуру – бінарне дерево пошуку, реалізує шість функцій: прямий обхід по вузлам дерева, симетричний обхід, обереneий обхід, обхід по рівням дерева, обчислення суми ключів лівих вузлів дерева, видалення дерева.