ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής

Εργασία Μαθήματος ***Δομές Δεδομένων***

Αριθμός εργασίας – Τίτλος εργασίας	***Υλοποίηση BinarySearchTree(BST)***
Όνομα φοιτητή	Κωνσταντίνος Καλογερόπουλος Μιχάλης Στυλιανίδης
Αρ. Μητρώου	Π19057/Π19165
Ημερομηνία παράδοσης	15/7/20

1 Εισαγωγή

Στόχος της εργασίας είναι η υλοποίηση μιας ουράς προτεραιότητας με την χρήση Binary Search Tree (BST).

2 Περιγραφή του προγράμματος

Υλοποιεί τις βασικές λειτουργίες της εισαγωγής κόμβων, της διαγράφης κόμβων, της εύρεσης μέγιστου, της εύρεσης ελάχιστου και της αναζήτησης κόμβων.

3 Επίδειξη της λύσης

```
//δημιουργια binary tree
struct BtNode {
    int data; // μεταβλητη που περιεχει το στοιχειο του κομβου του δεντρου
    BtNode* left; // pointer που δειχνει στο αριστερο παιδι
    BtNode* right; // pointer που δειχνει στο δεξι παιδι
};
```

Εικόνα 1 Το struct ενός κόμβου

```
BtNode* getNewNode(int data){
    BtNode* newNode = new BtNode(); //δεσμευουμε την
    //θα μπορουοσαμε να χρησιμοποιήσουμε και την mal
    newNode->data = data; //εισαγουμε στην νεα δεσμε
    newNode->left = newNode->right = NULL;//οριζουμε
    return newNode;
}
```

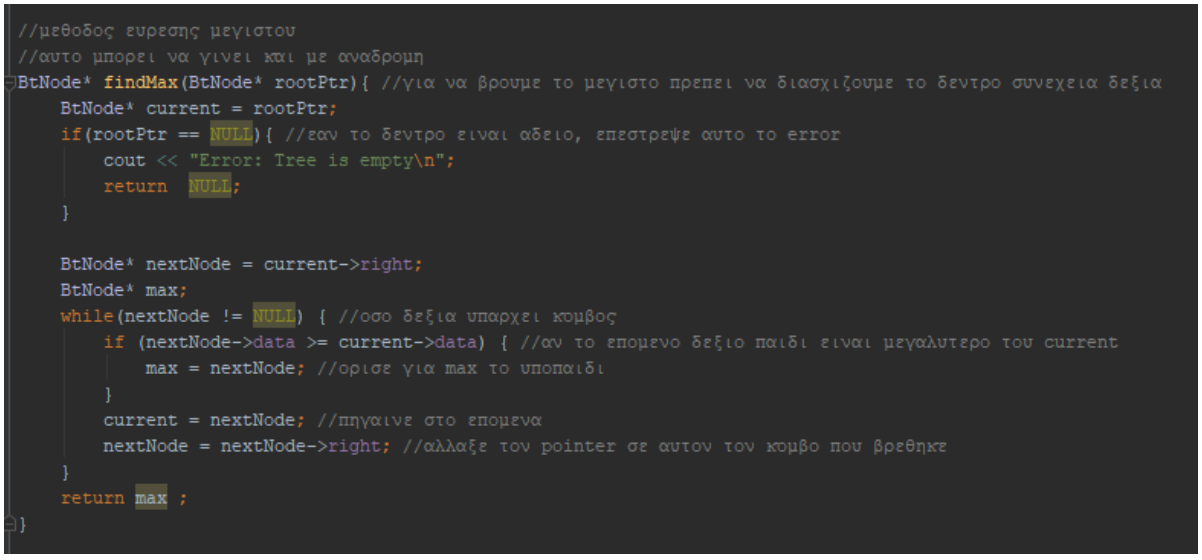
Εικόνα 2 Δημιουργία νέου κόμβου

```
//Function to find minimum in a tree.
BtNode* findMin(BtNode *root)
{
    while(root->left != NULL) root = root->left;
    return root;
}
```

Εικόνα 3 Συνάρτηση εύρεσης ελάχιστού

```
BtNode* deleteElement(BtNode *rootPtr, int data) {
    if(rootPtr == NULL) return rootPtr;
    else if(data < rootPtr->data) rootPtr->left = deleteElement(rootPtr->left, data);
    else if (data > rootPtr->data) rootPtr->right = deleteElement(rootPtr->right, data);
    else {
            // Case 1:  Δεν εχει παιδια
        if(rootPtr->left == NULL && rootPtr->right == NULL) {
            delete rootPtr;
            rootPtr = NULL;
        }
            //Case 2: Ενα παιδι
        else if(rootPtr->left == NULL) {
            BtNode *temp = rootPtr;
            rootPtr = rootPtr->right;
            delete temp;
        }
        else if(rootPtr->right == NULL) {
            BtNode *temp = rootPtr;
            rootPtr = rootPtr->left;
            delete temp;
        }
            // case 3: 2 παιδιά
        else {
            BtNode  *temp = findMin(rootPtr->right);
            rootPtr->data = temp->data;
            rootPtr->right = deleteElement(rootPtr->right, temp->data);
        }
    }
    return rootPtr;
}
```

Εικόνα 4 Συνάρτηση διαγράφης κόμβου



```
//μεθοδος ευρεσης μεγιστου
//αυτο μπορει να γινει και με αναδρομη
BtNode* findMax(BtNode* rootPtr){ //για να βρουμε το μεγιστο πρεπει να διασχιζουμε το δεντρο συνεχεια δεξια
    BtNode* current = rootPtr;
    if(rootPtr == NULL){ //εαν το δεντρο ειναι αδειο, επεστρεψε αυτο το error
        cout << "Error: Tree is empty\n";
        return  NULL;
    }

    BtNode* nextNode = current->right;
    BtNode* max;
    while(nextNode != NULL) { //οσο δεξια υπαρχει κομβος
        if (nextNode->data >= current->data) { //αν το επομενο δεξιο παιδι ειναι μεγαλυτερο του current
            max = nextNode; //ορισε για max το υποπαιδι
        }
        current = nextNode; //πηγαινε στο επομενα
        nextNode = nextNode->right; //αλλαξε τον pointer σε αυτον τον κομβο που βρεθηκε
    }
    return max ;
}
```

Εικόνα 5 Συνάρτηση εύρεσης μεγίστου

```
//μεθοδος για εισαγωγη κομβων στο δεντρο
BtNode* insert(BtNode* rootPtr, int data){
    if(rootPtr == NULL){
        rootPtr = getNewNode(data);
    }
    else if(data <= rootPtr->data){
        rootPtr->left = insert(rootPtr->left,data); //δημιουργια αριστερου παιδιου
    }
    else{
        rootPtr->right = insert(rootPtr->right,data); //δημιουργια δεξιου παιδιου
    }
    return rootPtr;
}
```

Εικόνα 6 Συνάρτηση εισαγωγής νέου κόμβου στο δέντρο

```
//μεθοδος αναζητησης κομβων
bool search(BtNode *rootPtr, int data){
    if(rootPtr == NULL) return  false;
    else if(rootPtr->data == data) return true; //
    else if(data <= rootPtr->data) return search(r
    else return search(rootPtr->right, data); // c
}
```

Εικόνα 7 Συνάρτηση αναζήτησης κόμβου με χρήση αναδρομής

```
int main() {

    BtNode* rootPtr; // pointer που δειχνει στην ρίζα του δεντρου
    rootPtr = NULL; //αρχικοποιουμαι το δεντρο στο κενο
    rootPtr = insert(rootPtr,  12);     rootPtr = insert(rootPtr,52);
    rootPtr = insert(rootPtr, 20);    rootPtr = insert(rootPtr,2);
    rootPtr = insert(rootPtr, 4);    rootPtr = insert(rootPtr, 100);
    rootPtr = insert(rootPtr, 10);    rootPtr = insert(rootPtr,245);
    rootPtr = insert(rootPtr, 5);    rootPtr = insert(rootPtr, 2);



    BtNode* maxNode = findMax(rootPtr); //ευρεση μεγιστου
    cout<<"The maximum is: " << maxNode->data<<endl;

    deleteElement(maxNode, maxNode->data); //διαγραφη μεγιστου
    cout <<"The maximum has been deleted\n";
    int  number;
    cout<<"Enter the number you want to search in the binary tree"<<endl;
    cin>>number;
    if(search(rootPtr, number) == true) cout <<"found\n";
    else cout<<"Not found\n";


}
```

Εικόνα 8 Δημιουργία ενός δέντρου και εκτέλεση των παραπάνω λειτουργιών.