**עצים בינאריים binaryTrees-**

**עץ בינארי:** מבנה נתונים שמיוצג כך- עץ שבו לכל קודקוד יש לכל היותר שני בנים, ולכל קודקוד, פרט לקודקוד מסוים הנקרא שורש, אב יחיד.

**מימוש המבנה בקוד:** רוב מימוש המבנה יהיה באמצעות רקורסיה ניתן לעשות גם שימוש בקוד ללארקורסיה אך יש להיעזר בתור.

**פונקציות שימושיות:**

**void AddLeaf(TLeaf\* curr, int val)**

**void printTree(TLeaf\* leaf)**

**/////////////////////////////////**

**void Queue\_Push(TLeaf\* leaf)**

**int Queue\_IsEmpty()**

**void Queue\_Clear()**

**TQueueItem\* Queue\_Pop()**

**void AddTreeItem(DetailsT item)**

**TLeaf\* FindEmptyPlaceLeaf(TLeaf\* leaf)**

**void BuildTree()**

**מימוש בעזרת תור וללא רקורסיה:**

typedef struct

{

unsigned int age;

} DetailsT;

typedef struct Leaf

{

// Management or MetaData

int level;

struct Leaf\* Left;

struct Leaf\* Right;

struct Leaf\* Parent;

// Data (Real Data)

DetailsT Details;

} TLeaf;

struct Leaf\* Root;

typedef struct QueueItem {

TLeaf\* leaf;

int Id;

struct QueueItem\* prev;

struct QueueItem\* next;

} TQueueItem;

TQueueItem\* Queue\_Head;

TQueueItem\* Queue\_Tail;

void Queue\_Push(TLeaf\* leaf)

{

//malloc

TQueueItem\* newItem = (TQueueItem\*)malloc(sizeof(TQueueItem));

newItem->leaf = leaf;

newItem->next = NULL;

if (Queue\_Head == NULL)

{

Queue\_Head = newItem;

Queue\_Tail = newItem;

newItem->next = NULL;

newItem->prev = NULL;

}

else

{

Queue\_Head->next = newItem;

newItem->prev = Queue\_Head;

Queue\_Head = newItem;

}

}

int Queue\_IsEmpty()

{

if (Queue\_Head == NULL)

{

return 1;

}

else

{

return 0;

}

}

void Queue\_Clear()

{

Queue\_Head = NULL;

Queue\_Tail = NULL;

}

TQueueItem\* Queue\_Pop()

{

TQueueItem\* ret = NULL;

if (Queue\_Head == NULL)

{

return NULL;

}

else if (Queue\_Head == Queue\_Tail)

{

ret = Queue\_Head;

Queue\_Head = NULL;

Queue\_Tail = NULL;

return ret;

}

else {

ret = Queue\_Tail;

Queue\_Tail->next->prev = NULL;

Queue\_Tail = Queue\_Tail->next;

return ret;

}

}

//////////////////////////////////////

void AddTreeItem(DetailsT item)

{

// Add new item into Tree

TLeaf\* newLeaf = (TLeaf\*)malloc(sizeof(TLeaf));

newLeaf->Details = item;

newLeaf->Right = NULL;

newLeaf->Left = NULL;

if (Root == NULL)

{

newLeaf->Parent = NULL;

Root = newLeaf;

newLeaf->level = 0;

}

else

{

TQueueItem\* curr;

TLeaf\* foundEmptyPlace = NULL;

Queue\_Push(Root);

while ((curr = Queue\_Pop()) != NULL)

{

foundEmptyPlace = NULL;

if (curr->leaf->Left == NULL)

{

foundEmptyPlace = curr->leaf;

foundEmptyPlace->Left = newLeaf;

newLeaf->Parent = foundEmptyPlace;

}

else if (curr->leaf->Right == NULL)

{

foundEmptyPlace = curr->leaf;

foundEmptyPlace->Right = newLeaf;

newLeaf->Parent = foundEmptyPlace;

}

else

{

Queue\_Push(curr->leaf->Left);

Queue\_Push(curr->leaf->Right);

}

if (foundEmptyPlace)

Queue\_Clear();

}

}

}

void BuildTree()

{

for (int i = 0; i < 100; i++)

{

DetailsT\* details = (DetailsT\*)malloc(sizeof(DetailsT));

details->age = i;

AddTreeItem(\*details);

}

}

TLeaf\* FindEmptyPlaceLeaf(TLeaf\* leaf)

{

if (leaf->Left != NULL)

return leaf->Left;

else if (leaf->Right != NULL)

return leaf->Right;

else

return NULL;

}

**מימוש בעזרת רקורסיה:**

typedef struct Leaf

{

// Management or MetaData

struct Leaf\* Left;

struct Leaf\* Right;

struct Leaf\* Parent;

// Data (Real Data)

int Val;

} TLeaf;

TLeaf\* Root;

void AddLeaf(TLeaf\* curr, int val)

{

TLeaf\* newLeaf = (TLeaf\*)malloc(sizeof(TLeaf));

newLeaf->Val = val;

newLeaf->Right = NULL;

newLeaf->Left = NULL;

if (Root == NULL)

{

Root = newLeaf;

newLeaf->Parent = NULL;

}

else {

if (curr == NULL)

{

curr = Root;

}

if (curr->Val < val)

{

if (curr->Right == NULL)

{

// add into

curr->Right = newLeaf;

newLeaf->Parent = curr;

}

else

{

AddLeaf(curr->Right, val);

}

}

else

{

if (curr->Left == NULL)

{

// add into

curr->Left = newLeaf;

newLeaf->Parent = curr;

}

else

{

AddLeaf(curr->Left, val);

}

}

}

}

void printTree(TLeaf\* leaf)

{

if (!leaf)

{

return;

}

printTree(leaf->Left);

printf("(%d)->", leaf->Val);

printTree(leaf->Right);

}