COMPUTER ENGINEERING LABORATORY

Luigi Rizzo

luigi.rizzo@unipd.it

October 2023-January 2024

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Lab exercises

Università degli Studi di Padova

Doubly linked lists

Problem: recording, retrieving, updating and saving Golf Scores by using a doubly linked list.

In recording scores for a golf tournament, we enter the name and score of the player as the player finishes. This information is to be retrieved in each of the following ways:

- Scores and names can be printed in order by ascending or by descending scores.
- Given the name of a player, other players with the same score shall be printed.

Doubly linked lists

The program will print the following menu

[1] Print list in ascending order of scores

[2] Print list in descending order of scores

[3] Search player

[4] Load new scores

[5] Save scores

[6] Exit

Make your choice:

Make use of the three files **scores*n*.txt** in order to update twice golf scores.

Print list in both orders and save data after first reading and after each update.

Binary search trees

Problem: recording the data of 20 worldwide capitals from the file capitals.txt in a binary search tree (ordered considering the population) and

- Print the tree in ascending order of population;
- Print the tree in descending order of population;
- Search for a capital and print its population
- Insert new capitals data from file capitals2.txt in the binary search tree, print the above mentioned lists and search for a capital.

Binary search trees

The program will print the following menu

[1] Print tree in ascending order of population

[2] Print tree in descending order of population

[3] Search capital

[4] Load new capitals

[5] Save data in ascending order of population

[6] Exit

Make your choice:

Insert operation


```
// insertion
struct node* insert(struct node * root, int x) {
     //searching for the place to insert
    if (root == NULL)
        return newNode(x);
    else if (x > root->data) // x is greater. Should be inserted to the right
        root->right_child = insert(root->right_child, x);
    else // x is smaller and should be inserted to left
        root->left_child = insert(root->left_child, x);
    return root;
}
```

Search operation


```
// searching operation
struct node* search(struct node * root, int x) {
    if (root == NULL || root->data == x) //if root->data is x the element is found
        return root;
    else if (x > root->data) // x is greater, so we will search the right subtree
        return search(root->right_child, x);
    else //x is smaller than the data, so we will search the left subtree
        return search(root->left_child, x);
}
```