Department of Computer Science

COS284 Practical and Assignment 6: Data Structures and Structs

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

1 Introduction

This document contains both Practical 6 and Assignment 6. In general the Assignments will build upon the work of the current Practical.

1.1 Submission

The Practical will be fitchfork only and is due at 20:00 on Tuesday the 15th of October. The Assignment will also be fitchfork only and is due at 17:00 on Friday 25th of October. You may use the practical sessions to work on your assignment in the labs and ask for assistance if it is required.

1.2 Plagiarism policy

It is in your own interest that you, at all times, act responsible and ethically. As with any work done for the purpose of your university degree, remember that the University of Pretoria will not tolerate plagiarism. Do not copy a friend's work or allow a friend to copy yours. Doing so constitutes plagiarism, and apart from not gaining the experience intended, you may face disciplinary action as a result.

For more on the University of Pretoria's plagiarism policy, you may visit the following webpage: `http://www.library.up.ac.za/plagiarism/index.htm`

1.3 Practical component [25%]

There are three tasks you must complete for this practical. All tasks will be marked by fitchfork. All tasks in this practical will involve working with the following struct,

```
struct Trie {
        struct Trie *children[26];
        char isWord[26];
};
```

1.3.1 Task 1: Allocating and Initialising

You must implement two functions called trieAlloc and trieInit in assembly. trieAlloc must only allocate and return memory for the Trie node struct. You must allocate the correct amount of memory including padding. The trieInit function must take in a pointer of type Trie and set all pointer and isWord values in the Trie to zero.

When you are finished, create a tarball with your assembly file "alloc.asm" and upload it to the practical 6 task 1 slot.

1.3.2 Task 2: Inserting

For this task you must implement the function trieInsert in assembly. The function will accept a pointer to the root node of the trie and a word as a string. You must subsequently insert the string into the trie. As follows,

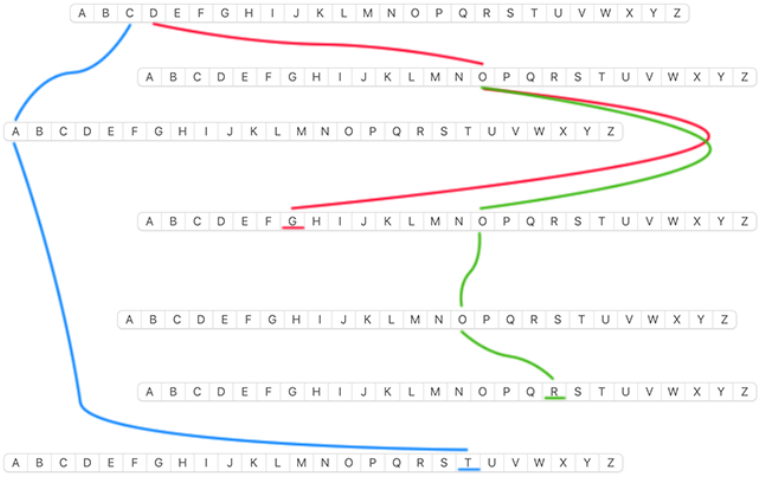
```
current = root

for each letter in word:
        let letterIndex = letter - 'a'
        if is last letter:
                current->isWord[letterIndex] = 1
        else:
                if needed create child at letterIndex
                current = current->children[letterIndex]
```

As an example if you insert the words "cat", "dog" and "door" into the trie you should have the following structure,

In the image the blue, red and green lines represents "cat", "dog" and "door" respectively. Note that "dog" and "door" share 2 nodes because they both have a prefix of "do" which has length 2.

When you are finished, create a tarball with your assembly files ("alloc.asm" and "insert.asm") and upload it to the practical 6 task 2 slot.



1.3.3 Task 3: Contains

In this task you must implement a function to check if your trie contains a given word. Your method will be used to solve a word search problem using brute force. The word search problem solution is provided for you in the given files. It is only your responsibility to implement the trieContains function.

Your contains function will accept the root of the trie and a string for which it must check. The trie given to the function will be a trie loaded with words from the "sowpods.txt" dictionary.

Your contains function should work as follows,

```
current = root

for each letter in word:
        let letterIndex = letter - 'a'
        if is last letter:
                return current->isWord[letterIndex] == 1
        else if current->children[letterIndex] == 0:
                return 0
        else:
                current = current->children[letterIndex]
return 0
```

When you are finished, create a tarball with your assembly files ("alloc.asm", "insert.asm" and "contains.asm") and upload it to the practical 6 task 3 slot.

1.4 Assignment component [75%]

There are three tasks you must complete for this assignment. All tasks will be marked by fitchfork. All tasks in this assignment will involve working with the following structs the board:

```
struct Board{
    struct Tile* tl;
    struct Tile* tm;
    struct Tile* tr;
    struct Tile* ml;
    struct Tile* mm;
    struct Tile* mr;
    struct Tile* bl;
    struct Tile* bm;
    struct Tile* br;
};
```

and the tile:

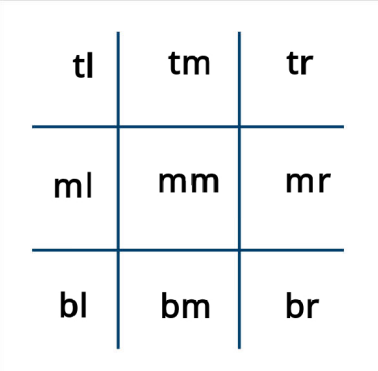
```
struct Tile{
        long player;
    struct Tile* nextTile;
};
```

1.4.1 Task 1: Generate board

You must implement a function called createBoard in assembly. createBoard must allocate and return memory for the Board struct. You must allocate the correct amount of memory including padding. The createBoard function should:

1. Allocate memory **for** nine Tile **struct** nodes.
2. Set the player value in each tile to 0.
3. Link each tile to the following tile, in a linked list manner, **using** nextTile.
3. Link each tile to the board **using** their indicated names.
4. Return a pointer to the board **struct**.

The structure of the board is as follows:

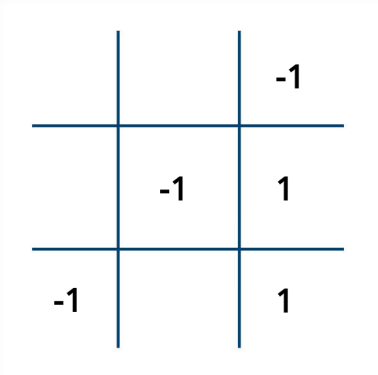


The board (linked list) should look as follows when correctly implemented:

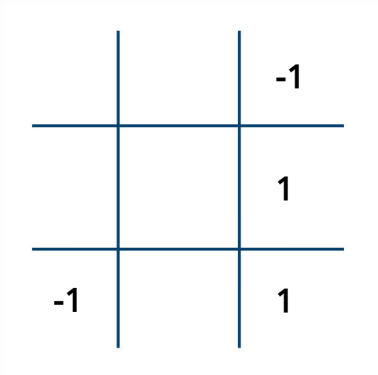
When you are finished, create a tarball with your assembly file "crtBrd.asm" and upload it to the assignment 6 task 1 slot.

1.4.2 Task 2: Determine winner

You must implement a function called win in assembly. win should take in the board struct and determine if a player has won. If a player has won it should return the player number who won (1 for player, -1 for computer) or 0 if no one has won yet.
The following board should return -1:



The following board should return 0:



When you are finished, create a tarball with your assembly files ("crtBrd.asm" and "detWin.asm") and upload it to the assignment 6 task 2 slot.

1.4.3 Task 3: Determine odds

You must implement a function called minmax in assembly. minmax should take in a board struct and the player number whose turn it is (1 or -1). It should first check if a player has won, if a player has won return a beneficial number (explained below) otherwise it should use a min-max tree to determine if the player can still win.

```
int winner = win(board);
// if their is a win return a beneficial number to the player,
```

```
//if the current player has won, return 1, else return -1
if(winner != 0) return winner*playerNum;

move = -1;
score = -2; //Losing moves are preferred to no move
for all tiles,
    if(tile->player == 0) { //If legal move,
        tile->player = playerNum //Try the move
        thisScore = - minmax(board, playerNum*-1)
        if(thisScore > score) {
            score = thisScore
            move = 1
        } //Pick the one that's worst for the opponent
        tile->player = 0 //Reset board after try
    }
}
if(move == -1) return 0
return score
```

When you are finished, create a tarball with your assembly files ("crtBrd.asm" , "detWin.asm" and "detOdds.asm") and upload it to the assignment 6 task 3 slot.

1.5 A note on the assignment tasks:

For all assignment tasks you will be given a C wrapper class that will handle input and output for you. You must simply implement the functions that will be called from inside the class and provide the correct return values.

2 Mark Distribution

Activity	**Mark**
Prac Task 1	5
Prac Task 2	10
Prac Task 3	10
Assignment Task 1	10
Assignment Task 2	25
Assignment Task 3	40
Total	**100**