Connor McCullough

EEN 521

Project 1 – Linked Lists and Binary Trees in BCPL

Results:

```
20:17 ~ > run treemachine
1
7
5
8
9
2
3
5
4
7
9
8
4
6
-1

 List:
1 7 5 8 9 2 3 5 4 7 9 8 4 6
 Tree:
1 2 3 4 4 5 5 6 7 7 8 8 9 9

456456
234
5786
1
4656
24
68567
435
976
345
134
125356
34257
456
238
36
-1

 List:
456456 234 5786 1 4656 24 68567 435 976 345 134 125356 34257 456 238 36
 Tree:
1 24 36 134 234 238 345 435 456 976 4656 5786 34257 68567 125356 456456
```

Code:

```
import "io"

manifest
{
        node_data = 0,
        node_left = 1,
        node_right = 2,
        sizeof_node = 3,
        link_data = 0,
        link_next = 1,
        sizeof_link = 2,
        heap_size = 10000
}

let new_node(x) be
{
        let p = newvec(sizeof_node);
        p ! node_data := x;
        p ! node_left := nil;
        p ! node_right := nil;
        resultis p
}

let new_link(x) be
{
        let p = newvec(sizeof_link);
        p ! link_data := x;
        p ! link_next := nil;
        resultis p
}

let add_to_list(ptr,value) be
{
        if ptr = nil then
                resultis new_link(value);
        ptr ! link_next := add_to_list(ptr ! link_next,value);
        resultis ptr
}

let add_to_tree(ptr, value) be
{
         if ptr = nil then
         resultis new_node(value);
         test value < ptr ! node_data then
         ptr ! node_left := add_to_tree(ptr ! node_left,
     value)
         else
              ptr ! node_right := add_to_tree(ptr ! node_right,
         value);
         resultis ptr
}
```

```
let list_to_tree(link) be
{
        let tree = nil;
        if link = nil then return;
        until link = nil do
                {
                tree := add_to_tree(tree,link ! link_data);
                link := link ! link_next;
                }
        resultis tree
}

let inorder_print(ptr) be
{
         if ptr = nil then return;
         inorder_print(ptr ! node_left);
         out("%d ", ptr ! node_data);
         inorder_print(ptr ! node_right)
}

let list_print(ptr) be
{
        if ptr = nil then return;
        out("%d ", ptr ! link_data);
        list_print(ptr ! link_next);
}

let start() be
{
        let inData;
        let heap = vec(heap_size);
        let link = nil;
        let tree = nil;
        init(heap,heap_size);

        until inData = -1 do
        {
                inData := inno();
                if inData = -1 then break;
                link := add_to_list(link,inData);
        }

        list_print(link);
        tree := list_to_tree(link);
         out("\n Tree: \n");
        inorder_print(tree);
}
```