DS Assignment IV

1. Suppose a singly linked linear list is created with integer data and the beginning of the list is available in a pointer ‘head’. Note that there is no distinct header node in the list and the first node itself contains relevant data. Write the following functions:
   A. Write a function to remove duplicates which takes a sorted list in increasing order pointed by head and deletes any duplicate nodes from the list. However, the list should only be traversed once.
   B. Write a function to release a list that takes a pointer ‘head’ pointing to the beginning of the list. The program should deallocate all the nodes and set its head pointer to NULL.

```c
typedef struct Node {
    int data;
    struct Node *next;
} NODE_t;

typedef NODE_t * NODE_p_t;



void removeDuplicates (NODE_p_t list) {
    NODE_p_t p = list;
    while (p != NULL) {
     NODE_p_t temp = p;
     while (temp != NULL && temp->data == p->data) {
           p->next = temp->next;
           temp = temp->next;
     }
     p = p->next;
    }
}


void freeMemory (NODE_p_t list) {
    NODE_p_t p = list;
    while (p != list) {
     NODE_p_t temp = p;
     p = p->next;
     free(temp);
    }
    list = NULL;
}
```

2. Considering **array representation** of a binary tree write the functions for the following:
   i) Insert an element into a binary tree iteratively. First insertion is the root of the tree. Following insertion should ask the user to input a choice as either left child (1) or a right child (2) as long as the space is occupied. If an empty space is available the element should be inserted.
   ii) Functions for inorder, preorder traversal .
   iii) Write a main function which provides a menu driven choice menu.

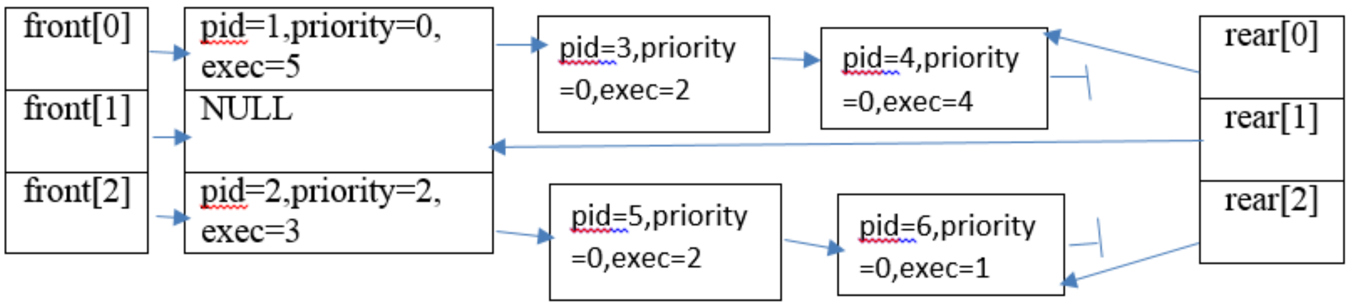
```
//
//    BinaryTree.c
//
//    Created by Avikant Saini on 10/19/15
//

/**
 Sample input : 1 10 1 8 1 1 3 1 1 1 5 1 2 1 2 2 1 2 2 1 for
       10
      /   \
    8      2
   /  \    /
 3    5  2
 Inorder:  3 8 5 10 2 2
 Preorder:  10 8 3 5 2 2
 */

#include <stdio.h>
#include <stdlib.h>

#define SIZE 100

typedef enum { NO, YES } BOOL;

typedef struct tnode {
     int data;
     struct tnode *left;
     struct tnode *right;
} TNODE_t;

typedef TNODE_t * TNODE_p_t;

// Stack crap

typedef struct stack {
     TNODE_p_t arr[SIZE];
     int tos;
} STACK_t;

typedef STACK_t * STACK_p_t;

void push (STACK_p_t stack, TNODE_p_t item) {
     if (stack->tos == SIZE - 1) {
       printf("OVERFLOW!");
       return;
     }
     stack->arr[++(stack->tos)] = item;
}

TNODE_p_t pop (STACK_p_t stack) {
     if (stack->tos == -1)
       return NULL;
     return stack->arr[(stack->tos)--];
}
```

```c
// Tree creation

TNODE_p_t createLeaf(char data) {
     TNODE_p_t leaf = (TNODE_p_t)malloc(sizeof(TNODE_t));
     leaf->right = NULL;
     leaf->left = NULL;
     leaf->data = data;
     return leaf;
}

void insertIterative (TNODE_p_t *rt, int item) {

     int ch;
     TNODE_p_t t = *rt;
     TNODE_p_t present = NULL;
     TNODE_p_t newnode;

     if (*rt == NULL) {
       *rt = createLeaf(item);
       return;
     }

     do {
       printf("\tCurrent: '%d' | 1. Left%s; 2. Right%s | Choice: ", t->data, ((t->left
== NULL)?"(*)":""), ((t->right == NULL)?"(*)":""));
       scanf("%d", &ch);
       present = t;

       if (ch == 1)
             t= t->left;
       else if (ch == 2)
             t = t->right;

       if (t == NULL) {
             newnode = createLeaf(item);
             (ch == 1)?(present->left=newnode):(present->right=newnode);
             return;
       }

     } while (YES);
}

// Transversals

void inOrder (TNODE_p_t root) {

     TNODE_p_t current = root;

     STACK_t stack;
     stack.tos = -1;

     if (current == NULL)
       return;

     while (YES) {

       if (current !=  NULL) {
             push(&stack, current);
             current = current->left;
       }

       else {
             if (stack.tos > -1) {
                   current = pop(&stack);
                   printf(" %d", current->data);
                   current = current->right;
             }
```

```c
            else
                break;
        }
    }

}

void preOrder (TNODE_p_t root) {

    TNODE_p_t current = root;

    STACK_t stack;
    stack.tos = -1;

    if (current == NULL)
      return;

    push(&stack, current);

    while (stack.tos > -1) {

      TNODE_p_t temp = pop(&stack);
      printf(" %d", temp->data);

      if (temp->right)
            push(&stack, temp->right);
      if (temp->left)
            push(&stack, temp->left);

    }

}

int main (int argc, const char * argv[]) {

    int ch, el;
    TNODE_p_t root;
    root = NULL;

    while (1) {
      printf("\n\t1. Insert\n\t2. Inorder\n\t3. Preorder\n\tChoice: ");
      scanf(" %d", &ch);
      switch(ch) {
            case 1:
                if (root == NULL)
                    printf("\tEnter root value: ");
                else
                    printf("\tEnter value: ");
                scanf(" %d", &el);
                insertIterative(&root, el);
                break;
            case 2:
                inOrder(root);
                break;
            case 3:
                preOrder(root);
                break;
            default:
                exit(0);
      }
    }
}
```

3. Implement multilevel scheduling of processes with multiple queues using linked list. Create a structure Mqueue with variables process_id(int), priority(int), exection_time(int) and link to the next node. Create an array of pointers of type Mqueue for front and rear of max_size(predefined)
   i) Write a function void InputProcess(Mqueue *front[MAX],Mqueue *rear[MAX],int pid, int priority,int exec_time)
      The function should create a node dynamically and insert into queue number of a given priority. Eg. If the process has priority 3 then it will be added to 3rd queue. If the priority is greater than the max_queues needs to handled.
   ii) Write a function ScheduleProcess(Mqueue *front[MAX]). It executes a process from low priority queue(start from priority=0) to MAX priority queue. Once a process has finished executing, it is deleted from the queue and its finishing time is printed starting from timestamp 0. Consider the eg and sample output of the ScheduleProcess expected.

      Consider multiple queue of length 3. and process with its ID and priority and execution time



For the given multiple queue the evaluation function output would be as follows:
Executed pid1, ending time = 5
Executed pid3, ending time = 7
Executed pid4, ending time = 11
Executed pid2, ending time = 14
Executed pid5, ending time = 16
Executed pid6, ending time = 17

```c
//
//  ProcessScheduling.c
//  Implementing multilevel scheduling of processes with multiple queues using linked list.
//
//  Created by Avikant Saini on 10/18/15.
//  Copyright © 2015 avikantz. All rights reserved.
//

#include <stdio.h>
#include <stdlib.h>

int MAX = 50;

typedef struct MQueue {
    int process_id;
    int priority;
    int execution_time;
    struct MQueue *link;
} MQUEUE_t;

typedef MQUEUE_t * MQUEUE_p_t;
```

```c
MQUEUE_p_t createProcess (int pid, int priority, int exec_time) {
    MQUEUE_p_t process = (MQUEUE_p_t)malloc(sizeof(MQUEUE_t));
    process->process_id = pid;
    process->priority = priority;
    process->execution_time = exec_time;
    process->link = NULL;
    return process;
}

void initQueue (MQUEUE_p_t front[MAX], MQUEUE_p_t rear[MAX]) {
    int i;
    for (i = 0; i < MAX; ++i) {
      *(front + i) = createProcess(0, 0, 0);
      *(rear + i) = createProcess(0, 0, 0);
    }
}

void inputProcess (MQUEUE_p_t front[MAX], MQUEUE_p_t rear[MAX], int pid, int
priority, int exec_time) {

    if (priority > MAX) {
      int i = MAX;
      MAX *= 2;
      front = (MQUEUE_p_t *)realloc(front, MAX * sizeof(MQUEUE_p_t));
      rear = (MQUEUE_p_t *)realloc(front, MAX * sizeof(MQUEUE_p_t));
      for (; i < MAX; ++i) {
           *(front + i) = createProcess(0, 0, 0);
           *(rear + i) = createProcess(0, 0, 0);
      }
    }

    MQUEUE_p_t f = *(front + priority - 1);
    MQUEUE_p_t r = *(rear + priority - 1);

    MQUEUE_p_t process = createProcess(pid, priority, exec_time);
    process->link = NULL;

    if (f->link == NULL) {
      f->link = process;
      r->link = process;
    }

    else {
      MQUEUE_p_t temp = f;
      while (temp->link != NULL)
           temp = temp->link;
      temp->link = process;
      r = process;
    }
    return;
}
```

```
void scheduleProcess (MQUEUE_p_t front[MAX]) {
    int i;
    int timeStamp = 0;
    for (i = 0; i < MAX; ++i) {

      MQUEUE_p_t f = *(front + i);
      MQUEUE_p_t temp = f->link;

      while (temp != NULL) {
           timeStamp += temp->execution_time;
           printf("Executed pid%d, ending time = %d\n", temp->process_id,
timeStamp);
           temp = temp->link;
           f = f->link;
      }
    }
}
```

4. **Consider the implementation of multiple stacks with linked list. Consider the below structure for stack and top**
   struct element { int val; struct element *next};
   struct top{ struct element *node; struct top *link;};

   Initialise top_list(a pointer of top) to null

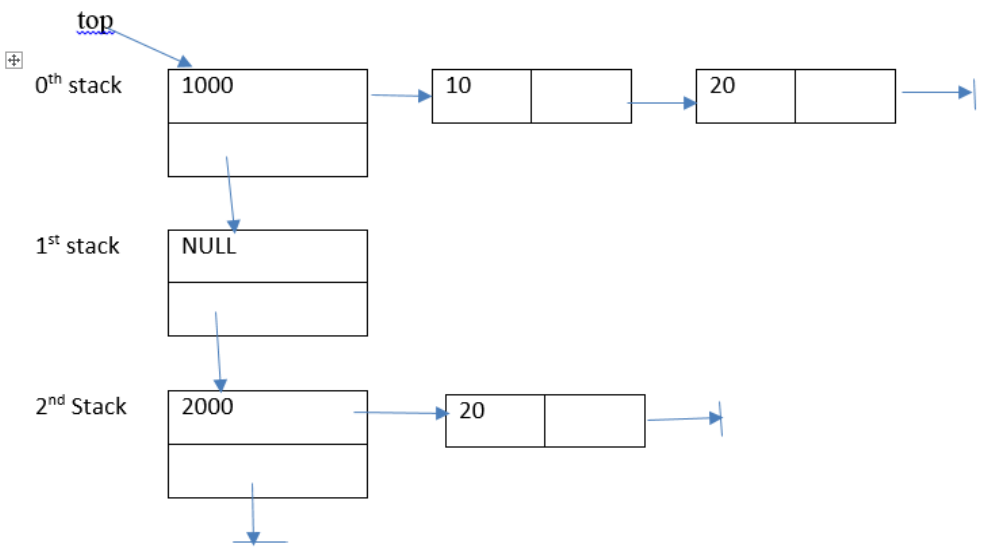
   i) Write a function BuildList(struct top **toplist, int max) that create a toplist to store max number of stacks. Assign the node as null and link as the next top in the list. The last top's link is set to null

   ii) Write a function push(struct top **toplist,int data, int i) which pushes data into the ith stack. Write a check if the stack number exceeds the max size.

   iii) Write a function pop(struct top **toplist,int i) which pops the top element from the top of the stack. Handle overflow and underflow conditions.

   iv) Write a main to show the usage of these functions.

   Eg: 3 stacks and elements



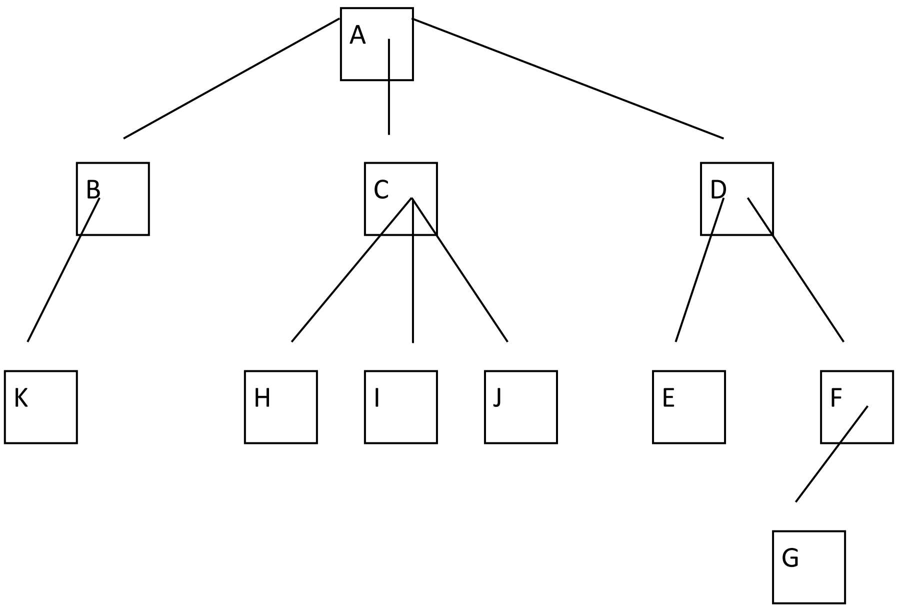
```c
//
//  MultipleStacksWithLinkedList.c
//  Implementation of multiple stacks with linked list.
//
//  Created by Avikant Saini on 10/18/15.
//  Copyright © 2015 avikantz. All rights reserved.
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define UNDERFLOW_INT -32767

typedef struct Element {
    int value;
    struct Element *next;
} ELEMENT_t;

typedef ELEMENT_t * ELEMENT_p_t;

ELEMENT_p_t createElement (int data) {
    ELEMENT_p_t element = (ELEMENT_p_t)malloc(sizeof(ELEMENT_t));
    element->value = data;
    element->next = NULL;
    return element;
}

typedef struct Top {
    ELEMENT_p_t node;
    struct Top *link;
} TOP_t;

typedef TOP_t * TOP_p_t;

void BuildList (TOP_p_t *topList, int max) {
    int i;

    for (i = 0; i < max; ++i) {
        *(topList + i) = (TOP_p_t)malloc(sizeof(TOP_t));
        TOP_p_t topi = *(topList + i);
        topi->node = NULL;
    }

    for (i = 0; i < max - 1; ++i) {
        TOP_p_t topi = *(topList + i);
        topi->link = *(topList + i + 1);
    }
    (*(topList + i + 1))->link = NULL;
}
```

```c
void push (TOP_p_t *topList, int data, int i) {
    TOP_p_t list = *(topList + i - 1);

    ELEMENT_p_t element = createElement(data);

    if (list->node == NULL)
           list->node = element;
    else {
           ELEMENT_p_t temp = list->node;
           while (temp->next != NULL)
                temp = temp->next;
           temp->next = element;
     }
}

int pop (TOP_p_t *topList, int i) {
    TOP_p_t list = *(topList + i - 1);
    // Base case (no element)
    if (list->node == NULL) {
           printf("\nList %d Underflow.\n\n", i);
           return UNDERFLOW_INT;
    }

    ELEMENT_p_t p = list->node;
    // Base case (one element)
    if (p->next == NULL) {
           int data = p->value;
           list->node = NULL;
           return data;
    }
    ELEMENT_p_t element = p;
    while (p->next != NULL) {
           element = p;
           p = p->next;
    }

    int data = p->value;
    element->next = NULL;

    return data;
}
```

```
int main (int argc, const char * argv []) {

    TOP_p_t * list = (TOP_p_t *)calloc(4, sizeof(TOP_p_t));
    BuildList(list, 4);

    push(list, 12, 1);
    push(list, 13, 1);
    push(list, 14, 1);          // List at index 1 now contains: 12 -> 13 -> 14

    push(list, 11, 2);
    push(list, 12, 2);
    push(list, 13, 2);          // List at index 2 now contains: 11 -> 12 -> 13

    push(list, 14, 3);
    push(list, 15, 3);          // List at index 3 now contains: 14 -> 15

    printf("\n%d", pop(list, 1));           // 14
    printf("\n%d", pop(list, 1));           // 13
    printf("\n%d\n", pop(list, 1));         // 12

    printf("\n%d", pop(list, 2));           // 13
    printf("\n%d\n", pop(list, 2));         // 12

    printf("\n%d", pop(list, 3));           // 15
    printf("\n%d", pop(list, 3));           // 14
    printf("\n%d\n", pop(list, 3));         // List 3 Underflow.

    printf("\n%d\n", pop(list, 4));         // List 4 Underflow.

}
```

Q5. i) What is a general tree? Give an algorithm to convert tree to binary tree.

ii) Using the same convert the below given tree to binary tree.

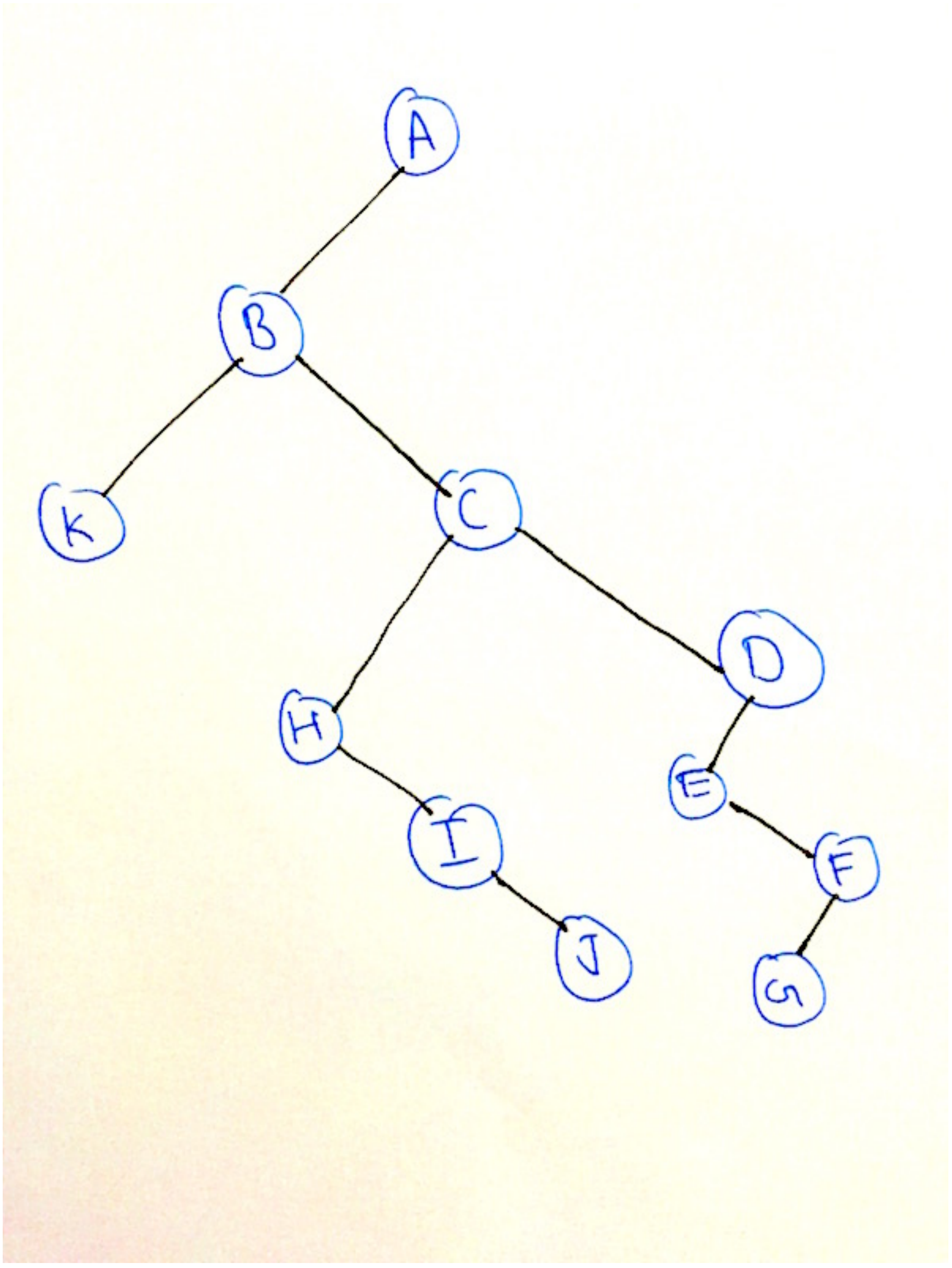


A general tree is a **data structure** in that each node can have infinite number of children, It cannot be empty, and there’s no limit on the degree of node.

GENERAL TREE to BINARY TREE

1. The first child of the general tree becomes the left child of the parent.

2. The subsequent children become the right child of their predecessor.


A
B
K
C
H
D
I
E
F
J
G