Assignment 1 V1.0
Session 08

1. **[20%]** What are the differences and how to implement these terms below:
   a. **[5%] Array and Linked List**

   Answer:

   Differences:

   Array is collection of same data type elements.

   Linked list is collection of same data type elements in which one element is connected to next element using pointers.

   Implementation of array:

   ```
   int a[5] = {1, 2, 3};
   ```

   Implementation of linked list:

   ```
   struct Node {
       int data;
       struct Node *next;
   }
   ```

   b. **[5%] Single and Double Linked List**

   Answer:

   Differences:

   Single linked list each node containing data and a pointer to next node.

   Double linked list contains nodes with each node containing data and a pointer to next node and a pointer to previous node.

   Implementation of single linked list:

   ```
   struct Node {
     int data;
     struct Node *next;
   }
   ```

   Implementation of double linked list:

   ```
   struct Node {
     int data;
     struct Node *next, *previous;
   }
   ```

Assignment 1 V1.0
Session 08

c. [5%] Stack and Queue

Answer:

Differences:

A stack is a data structure which elements can be inserted and deleted only from top side.

A queue is a data structure in which elements can be inserted only from rear side and deleted only from front side.

Implementation of stack:

```
stack <int> a;
a.push(data); //adding element to stack at top
a.pop(); //deleting element of stack from top
```

Implementation of queue:

```
queue <int> a;
a.push(data); //adding element from rear side
a.pop(); //deleting element from front side
```

d. [5%] Static and dynamic memory allocation

Answer:

Differences:

Static memory allocation is when variables are allocated memory before program execution.

Dynamic memory allocation is when variables are allocated memory after program execution during run time of program.

Implementation of static memory allocation:

```
int main(){
        int a, b; //static memory allocation
}
```

Implementation of dynamic memory allocation:

```
int main(){
        int *ptr = new int; //dynamic memory allocation
}
```

Assignment 1 V1.0
Session 08

2. **[30%]** Convert to **postfix** and **prefix** notation from expression below by using:

   a. **[15%] Stack Simulation**

   Answer:

   Prefix notation of A % B / C + (D + E * F – G) * H - I:

   I) Reverse the given infix expression

   Result ➔ I – H *) G – F * E + D (+ C / B % A

   II) Find postfix notation and reverse it which is the infix notation of given expression

Stack	Character	Prefix String
EMPTY	I	I
-	-	I
-	H	IH
- *	*	IH
- *)	)	IH
- *)	G	IHG
- *) -	-	IHG
- *) -	F	IHGF
- *) - *	*	IHGF
- *) - *	E	IHGFE
- *) - +	+	IHGFE*
- *) - +	D	IHGFE*D
- *	(	IHGFE*D+-
- +	+	IHGFE*D+-*
- +	C	IHGFE*D+-*C
- + /	/	IHGFE*D+-*C
- + /	B	IHGFE*D+-*CB
- + / %	%	IHGFE*D+-*CB
- + / %	A	IHGFE*D+-*CBA
		IHGFE*D+-*CBA%/+-

So, calculated string is "IHGFE*D+-*CBA%/+-". Hence REVERSE of this string is the prefix expression.

Prefix Expression: -+/%ABC*-+D*EFGHI

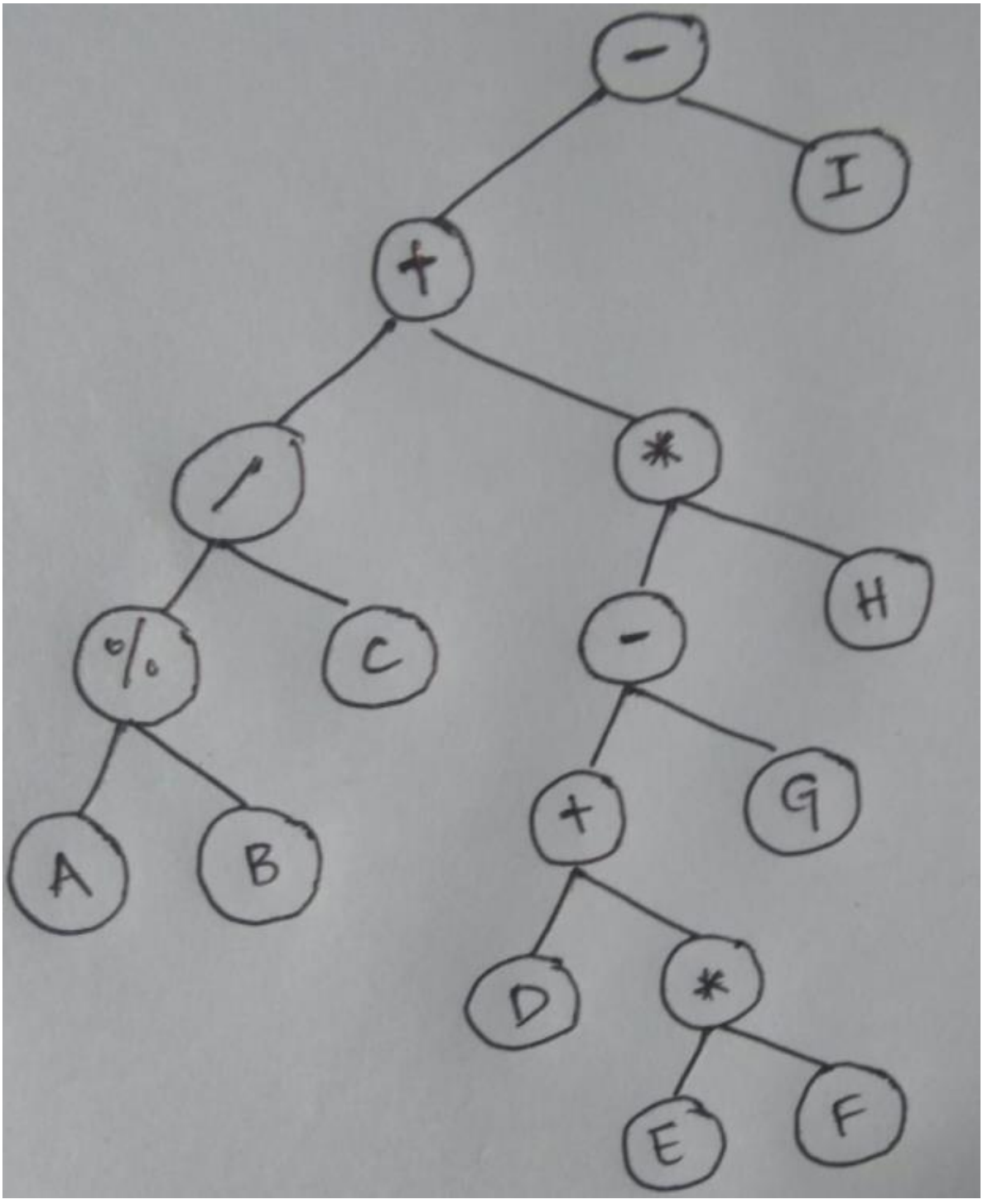
Postfix notation of A % B / C + (D + E * F - G) * H - I:

Stack	Character	Postfix String
EMPTY	A	A
%	%	A
%	B	AB
/	/	AB%
/	C	AB%C
+	+	AB%C/
+ (	(	AB%C/
+ (	D	AB%C/D
+ (+	+	AB%C/D
+ (+	E	AB%C/DE
+ (+ *	*	AB%C/DE
+ (+ *	F	AB%C/DEF
+ (-	-	AB%C/DEF*+
+ (-	G	AB%C/DEF*+G
+	)	AB%C/DEF*+G-
+ *	*	AB%C/DEF*+G-
+ *	H	AB%C/DEF*+G-H
-	-	AB%C/DEF*+G-H*+
-	I	AB%C/DEF*+G-H*+I
EMPTY		AB%C/DEF*+G-H*+I-

Postfix Expression: AB%C/DEF*+G-H*+I-

b. [15%] Expression Tree

Answer:



Prefix Notation: - + / % A B C * - + D * E F G H I

Postfix Notation: A B % C / D E F * + G - H * + I -

Assignment 1 V1.0
Session 08

A % B / C + (D + E * F – G) * H – I

Table 1 Precedence Order

Operator	Precedence	Associativity
*	2	left to right
/	2	left to right
%	2	left to right
+	1	left to right
-	1	left to right

Table 2 Stack Simulation

String	Stack	Prefix | Postfix String

Assignment 1 V1.0
Session 08

3. **[50%]** Code Snippet

Given snippet code below that you are required to complete. You are not allowed to make a new function or change any given code. Please complete each section that are marked with the notation “INSERT YOUR CODE HERE”.

Once you complete the snippet below, your output should have the same result with the given output below.

Descriptions:

a. **[15%]** isValid()

This function is for checking that there is no duplicated employee data in the linked list. This function will check the name, jobPos, grade and age. It will return to 1 if no duplicated data found at those properties and return to 0 vice versa.

b. **[10%]** push()

This function is built for inserting a new data in the linked list. You are required to check whether the inputted is novel data. If the inputted data is not valid, the program will print “ The inputted data is duplicated!”.

Note : please analyze how to push a data by given sequence of inserting in the main function and please compare it to the given output!

c. **[15%]** pop()

this function is built for deleting a data. If you are trying to delete a data when the list is empty, then the program will prompt you “the list is empty!”.

Note : please analyze how to pop a data by given sequence of deleting in the main function and please compare it to the given output!

d. **[10%]** printAll()

this function is built for printing all the data in the list along with the total employee number. If you are trying to delete a data when the list is empty, then the program will prompt you “the list is empty!”.

Assignment 1 V1.0
Session 08

```
//
// Created by Hanry Ham on 2020-05-24.
//
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

struct Employee{
    char name[20];
    char jobPos[15];
    int grade;
    int age;
    Employee *next;
}*head = NULL, *tail = NULL;

bool isValid(char *name, char *jobPos, int grade, int age){
    // [15%] (1) INSERT YOUR CODE HERE
    struct Employee *emp=head;
    while(emp!=NULL){
      if ((strcmp(name,emp->name)==0)&&
(strcmp(jobPos,emp->jobPos)==0)&&(grade==emp->grade)&&(age==emp->age)){
          return 0;
      }
      emp = emp->next;
   }
   return 1;
}

void push(char *name, char *jobPos, int grade, int age){
    struct Employee *curr = (struct Employee *) malloc(sizeof(Employee));
    // [10%] (2) INSERT YOUR CODE HERE
    if(isValid(name,jobPos,grade,age)){
        strcpy(curr->name,name);
        strcpy(curr->jobPos,jobPos);
        curr->grade = grade;
        curr->age = age;
        if (head==NULL || tail==NULL){
            curr->next = NULL;
            head=tail=curr;
        }else{
            curr->next = head;
            head = curr;
    }else{
```

```
            free(curr);
            printf("The inputted data is duplicated!\n");
        }
}
void pop(){
    struct Employee * curr = head;
    // [15%] (3) INSERT YOUR CODE HERE
      curr = tail;
      if(head==NULL || tail==NULL){
         printf("the list is empty!\n");
      }else{
          if(head==tail){
             head=tail=NULL;
          }else{
              struct Employee * tmp = head;
              while(tmp->next!=tail){
                    tmp=tmp->next;
              }
              tail = tmp;
              tail->next=NULL;
          }
          free(curr);
       }
}
void printAll(){
    printf("\n\n");
    struct Employee * curr = head;
    int empCtr = 0;
    if(!curr){
        printf("the list is empty!");
    }else{
        // [10%] (4) INSERT YOUR CODE HERE
printf("=====================================================\n");
printf("| Name| Job Position| Grade| Age|\n");
printf("=====================================================\n");
while(curr!=NULL){
      printf("| %s| %s| %d| %d|\n",curr->name,curr->jobPos,curr->grade,curr->age);
      empCtr++;
      curr = curr->next;
}
printf("=====================================================\n");
printf("Total Employee : %d\n",empCtr);
printf("=====================================================\n");
}
```

Assignment 1 V1.0
Session 08

```
int main(){
    pop();
    printAll();
    push("Hanry", "Supervisor", 12, 27);
    push("Yen", "Manager", 13, 40);
    pop();
    push("Derwin", "Manager", 15, 31);
    push("Andry", "Manager", 15, 30);
    pop();
    push("Saka", "Manager", 15, 32);
    pop();
    push("Afan", "Manager", 16, 35);
    push("Fredy", "Senior Manager", 18, 45);
    pop();
    printAll();
    return 0;
}
```

Output:

```
the list is empty


the list is empty!


=====================================================
|            Name|      Job Position| Grade|   Age|
=====================================================
|           Fredy|    Senior Manager|    18|    45|
|            Afan|           Manager|    16|    35|
|            Saka|           Manager|    15|    32|
=====================================================
Total Employee :  3
=====================================================
```

Assignment 1 V1.0
Session 08

OUTPUT CODE: