## EF234101 Fundamental Programming (F)

# Quiz 2

| | |
|---|---|
| Starting date: | 13 November 2024 |
| Deadline: | 20 November 2024, 23:59 WIB. **Penalty: 0.15% of grade/minute of tardiness.** |
| Exam type: | Open, Group (of at most three students) |
| Send to: | MM Irfan Subakti <yifana@gmail.com><br>CC to Sarah Nurhasna Khairunnisa <sarahnurhasna@gmail.com>, Widian Sasi Disertasiani <widianogik@gmail.com>, Arya Gading Prinandika <gprinandika@gmail.com> & Adnan Abdullah Juan <adnanjuan06@gmail.com> with the subject: **EF234101_FunPro(F)_Q2_StudentID1_Name1_StudentID2_Name2** |
| File type and format: | A **zip** file containing all of the **.c source files** & the **declaration(s)** |
| Filename format: | **EF234101_FunPro(F)_Q2_StudentID1_Name1_StudentID2_Name2**.ZIP |

**Instruction**
Please do these steps as in the following.

0. Please download the **template file**, extract it, and work based on those files.

1. Please update the program, namely `01_linked.c`. The purpose of this program is to sort a singly linked list in ascending order, i.e., at least 5 items in the linked list. You can use any sorting algorithm, e.g., Bubble sort, Selection sort, Merge sort, etc. but not the library-based sort (the function `sort()`). Once you have sorted the linked list, then insert an integer at the end of the linked list. [**33 points**]

```
#include <stdio.h>
#include <stdlib.h>
#define FALSE 0
#define TRUE 1 /* Any non-zero is true */

struct sNode {
      int data;
      struct sNode *next;
};

typedef struct sNode tNode;
typedef tNode *Node;
int flag = FALSE;
void insert(Node *start, int data);
void bubbleSort(Node start);
void swap(Node a, Node b);
void print(Node start);

int main() {
      int initArr[] = {10, 50, 15, 1, 100}; /* Initial value in
list, min 5 */
      Node start = NULL;
      for(int i = 0; i < 5; i++) {
            insert(&start, initArr[i]);
```

```
	}
	print(start);
	printf("Input: ");
	int input;
	scanf("%d", &input);
	insert(&start, input);
	bubbleSort(start);
	print(start);
	return 0;
}

void insert(Node *start, int data) {
	Node temp = malloc(sizeof(Node));
	// ----------------------------
	//
	//    Continue your code here
	//
	// ----------------------------
	*start = temp;
}

void print(Node start) {
	Node temp = start;
	while(temp != NULL) {
		if(temp->next != NULL) {
			printf("%d, ", temp->data);
	// ----------------------------
	//
	//    Continue your code here
	//
	// ----------------------------
		temp = temp->next;
	}
}

/* In this case we use Bubble Sort for the sorting algorithm */
void bubbleSort(Node start) {
	int swapped = 0;
	Node temp;
	Node temp1 = NULL;
	if(start == NULL) return;
	do {
		swapped = 0;
		temp = start;
		while(temp->next != temp1) {
	// ----------------------------
	//
	//    Continue your code here
	//
	// ----------------------------
		}
		temp1 = temp;
	}
	while(swapped);
	flag = TRUE;
}
```

```
void swap(Node a, Node b) {
      // ----------------------------
      //
      //    Continue your code here
      //
      // ----------------------------
}
```

**Input**
The integer number you want to insert into the linked list.

**Output before the next insertion**
The sorted elements are in ascending order.

**Input**
An integer.

**Final Output**
The sorted elements are in ascending order + the inserted integer.

```
100, 1, 15, 50, 10 -> before sorting
Input: 25
1, 10, 15, 25, 50, 100 -> after sorting
```

2. Update the program, namely `02_tree_from_file.c`. The purpose of this program is to create a binary tree from a file, namely `02_file_input.txt` as in the following. [**34 points**]
   a. First-line in that file will be the type of the node, there are three types of nodes:
      i. `0`: means the node is the root (the index will be started by 0, as the first node in the tree's hierarchy)
      ii. `l`: (small L) means the node must be inserted at the current's left node
      iii. `uuuur`: means the node must be inserted at the number of 'u' before this node's right node. In this example, '`uuuur`', there are 4 u's before 'r'.
   b. The next line is the data that needs to be inserted into the node

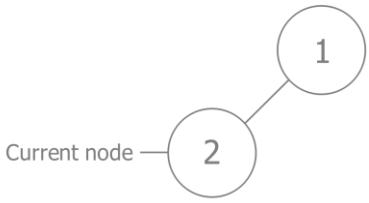
   **Example:**

   **The file (`02_file_input.txt`):**

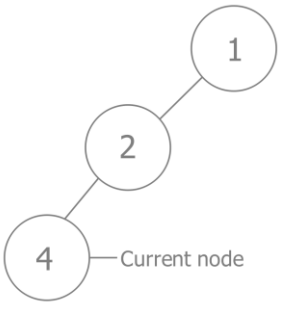
```
0
1
l
2
l
4
ur
5
uur
3
```

**The output will be:**
Line 1: `0`, this means it will be the root.
Line 2: `1` (number `1`), this means the node will have 1 as its data.
Tree:

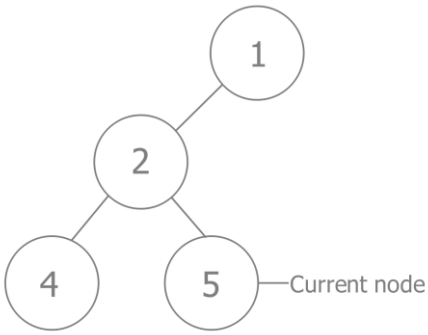

Line 3: `l` (small `l`); this means this node will be placed at the current node's left.
Line 4: `2`; the data is `2`.
Tree:



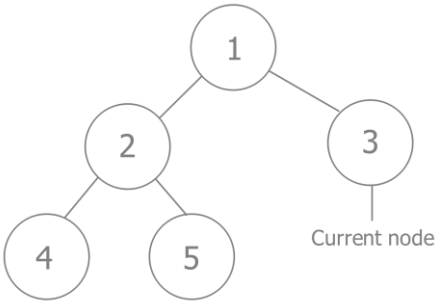

Line 5: `l` (small `l`); this means this node will be placed at the current node's left.
Line 6: `4`; the data is `4`.
Tree:



Line 7: `ur`; this means from the current node; we need to go to one upper node which is node 2 and put the current node on its right.
Line 8: `5`; the data is `5`.
Tree:

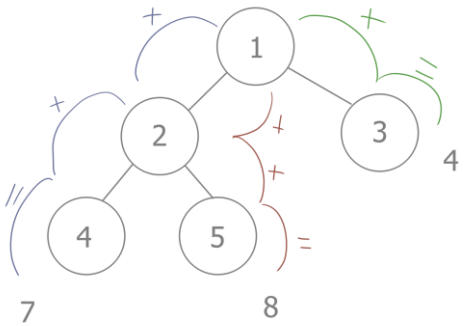



Line 9: `uur`; this means from the current node; we need to go to two upper (2 `u`'s) nodes which is node 1 and put the current node on its right.
Line 8: `3`; the data is `3`.
Tree:




c. The output will be the pre-order traversal of the built tree with the exception:
   i. It will print = node/leaf: data.
   ii. At the leaf instead, the data print the sum of the number to get to that leaf.

**Rough illustration:**




**Example:**

The output of the tree above will be:

```
node: 1
node: 2
leaf: 7
leaf: 8
leaf: 4
```

The third output is 7 because it is calculated from 1+2+4
The fourth output is 8 because it is calculated from 1+2+5
The fifth output is 4 because it is calculated from 1+3

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

struct treeNode {
   struct treeNode *left; /* treeNode pointer */
   int data; /* define data as an int */
   struct treeNode *right; /* treeNode pointer */
   struct treeNode *parent; /* treeNode pointer */
}; /* end structure treeNode */

typedef struct treeNode TreeNode;
typedef TreeNode *TreeNodePtr;

void preOrder(TreeNodePtr treePtr, int sum) {
   /* if tree is not empty then traverse */
   if ( treePtr->left != NULL || treePtr->right != NULL )
{ //is a node
         printf( "node: %3d\n", treePtr->data );
   // ----------------------------
   //
   //    Continue your code here
   //
   // ----------------------------
   } /* end if */
   else { //is a leaf
         printf("leaf : %d\n", sum+ treePtr->data);
   }
} /* end function preOrder */

TreeNodePtr insertLeft(TreeNodePtr from, int data) {
   TreeNodePtr d = malloc(sizeof(TreeNode));
   d->data = data;
   // ----------------------------
   //
   //    Continue your code here
   //
   // ----------------------------
   return d;
}
TreeNodePtr insertRight(TreeNodePtr from, int data) {
   TreeNodePtr d = malloc(sizeof(TreeNode));
   d->data = data;
```

```
    // ----------------------------
    //
    //    Continue your code here
    //
    // ----------------------------
    return d;
}
TreeNodePtr parseFile(char* filename, TreeNodePtr cur) {
    FILE *fptr;
    TreeNodePtr root = NULL;
    if ((fptr = fopen(filename,"r")) == NULL) {
         printf("Error! opening file");
         // Program exits if the file pointer returns NULL.
         exit(1);
    }
    char c[100] ;
    while (fgets(c, 100, (FILE*)fptr)) {
         printf("%s", c );
         char digit[5];

         fgets(digit, 5, (FILE*)fptr);
         for (int x = 0; x<5; x++) {
              if (digit[x]=='\n') digit[x] = '\0';
         }
         int data = atoi(digit);
         int count, cur2;
         if (c[0]=='l') {
              count = 0;
              cur2 = 2;
              while (1) { /* While (true) */
                   if (c[cur2] = '\n') {
                        break;
                   }
                   cur2+=1;
                   count+=1;
              }
              //get the digit here
              printf("left - data: %d\n",data);
              cur = insertLeft(cur, data);
         } else if (c[0]=='u') {
              cur2 = 1;
              while (c[cur2]=='u') {
                   cur2+=1;
              }
              for (int x = 0; x<cur2; x++) {
                   cur = cur->parent;
              }
              //get the digit here
              cur = insertRight(cur, data);
              //get the digit here
              printf("right - data: %d\n",data);
         } else if (c[0] == '0') {
              cur = malloc(sizeof(TreeNode));
              cur->parent = NULL;
              cur->left = NULL;
              cur->right = NULL;
```

```
                cur->parent = NULL;
                cur->data = data;
                count = 0;
                cur2 = 2;
                while (1) { /* while (true) */
                      if (c[cur2] = '\n') {
                            break;
                      }
                      cur2+=1;
                      count+=1;
                }
                //get the digit here
                root = cur;
                printf("root - data: %d\n",data);
          }
    }
    fclose(fptr);
    return root;
}

int main() {
    TreeNodePtr root = NULL;
    root = parseFile("02_file_input.txt", root);
    preOrder(root, 0);
    return 0;
}
```

```
0
root - data: 1
l
left - data: 2
l
left - data: 4
ur
right - data: 5
uur
right - data: 3
node:   1
node:   2
leaf : 7
leaf : 8
leaf : 4
```

3. Update the program, namely `03_chess_shortest_path.c`. The purpose of this program is to find the shortest path (minimum steps of the **horse/knight** piece on chess). Given an `8×8` chessboard, the minimum number of steps taken by a **horse/knight** to reach the *destination* from the current position. You may use any data structure for dynamic memory allocation which has been discussed in our class. [**33 points**]

   **Example**:
   **Input:**

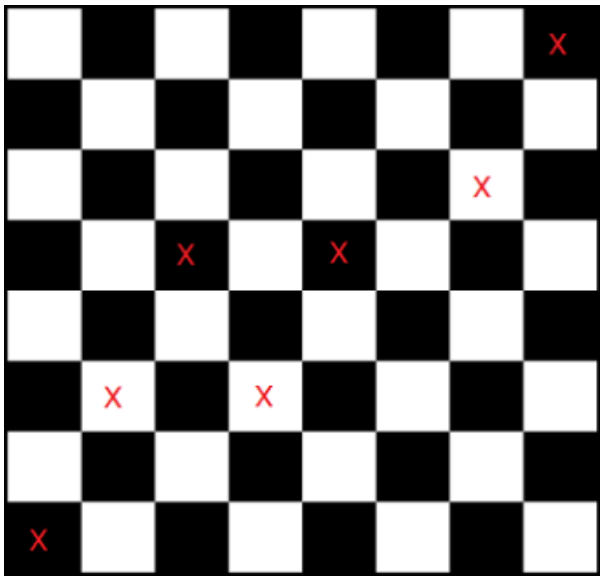
   ```
   Enter starting position (row col): 7 0
   Enter destination position (row col): 0 7
   ```

   Note: 7,0 is the *current position* and 0,7 is the *destination*.
   Remember the position started from 0, not 1.

   **Output:**

   ```
   Steps required: 6
   ```

   **Explanation:**



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

#define N 8

// Structure to represent a cell in the chessboard
typedef struct {
    int x, y;
    int distance;  // Stores the number of moves taken to
reach this cell
```

```
} Cell;

// Valid moves for a horse/knight on the chessboard
int rowMoves[] = {2, 2, -2, -2, 1, 1, -1, -1};
int colMoves[] = {-1, 1, -1, 1, 2, -2, 2, -2};

// Check if a cell is within the bounds of the chessboard
int isValid(int x, int y) {
    return (x >= 0 && y >= 0 && x < N && y < N);
}

// Queue node structure for BFS
typedef struct QueueNode {
    Cell cell;
    struct QueueNode *next;
} QueueNode;

// Queue data structure with head and tail pointers
typedef struct {
    QueueNode *head, *tail;
} Queue;

// Initialize an empty queue
void initQueue(Queue *queue) {
    queue->head = queue->tail = NULL;
}

// Enqueue a cell with its distance to the queue
void enqueue(Queue *queue, int x, int y, int distance) {
    QueueNode *newNode = (QueueNode
*)malloc(sizeof(QueueNode));
    newNode->cell.x = x;
    newNode->cell.y = y;
    newNode->cell.distance = distance;
    newNode->next = NULL;
    if (queue->tail) {
      // ----------------------------
      //
      //    Continue your code here
      //
      // ----------------------------

    } else {
      // ----------------------------
      //
      //    Continue your code here
      //
      // ----------------------------

    }
    queue->tail = newNode;
}

// Dequeue a cell from the front of the queue
Cell dequeue(Queue *queue) {
    QueueNode *temp = queue->head;
      // ----------------------------
      //
```

```
      //    Continue your code here
      //
      // ----------------------------
    free(temp);
    return cell;
}

// Check if the queue is empty
int isEmpty(Queue *queue) {
    return queue->head == NULL;
}

// BFS function to find the minimum steps for horse/knight to
reach the destination
int findMinSteps(int startX, int startY, int destX, int destY)
{
    // If starting position is the same as destination
    if (startX == destX && startY == destY) return 0;

    int visited[N][N] = {0};  // To keep track of visited
cells
    Queue queue;
    initQueue(&queue);

    // Enqueue the starting position with 0 distance
    enqueue(&queue, startX, startY, 0);
    visited[startX][startY] = 1;

    while (!isEmpty(&queue)) {
        // Get the front cell from the queue
        Cell cell = dequeue(&queue);

        // Check all possible moves for the knight
        for (int i = 0; i < 8; i++) {
            int newX = cell.x + rowMoves[i];
            int newY = cell.y + colMoves[i];

            // If the new position is valid and not visited
yet
            if (isValid(newX, newY) && !visited[newX][newY]) {
                // If the destination is reached, return the
distance
                if (newX == destX && newY == destY) {
                    return cell.distance + 1;
                }

                // Otherwise, enqueue the new position and
mark it as visited
                enqueue(&queue, newX, newY, cell.distance +
1);
                visited[newX][newY] = 1;
            }
        }
    }

    // Return a large value if destination is unreachable
(shouldn't happen on 8x8 board)
    return INT_MAX;
```

```
}

int main() {
    int startX, startY, destX, destY;

    // Input: starting and destination positions
    printf("Enter starting position (row col): ");
    scanf("%d %d", &startX, &startY);
    printf("Enter destination position (row col): ");
    scanf("%d %d", &destX, &destY);

    int steps = findMinSteps(startX, startY, destX, destY);
    printf("Steps required: %d\n", steps);

    return 0;
}
```

```
Enter starting position (row col): 7 0
Enter destination position (row col): 0 7
Steps required: 6
```

4. To avoid plagiarism/cheating, every student needs to pledge and declare, then she/he must submit her/his **signed pledge and declaration** as in the following. Failing to do so will result in getting a 0 (zero) grade. Attach the **scanned**/**photo** of your *declaration* in your report.

   "By the name of Allah (God) Almighty, herewith I pledge and truly declare that I have solved quiz 2 by myself, didn't do any cheating by any means, didn't do any plagiarism, and didn't accept anybody's help by any means. I am going to accept all of the consequences by any means if it has proven that I have done any cheating and/or plagiarism."

   [Place, e.g., Surabaya], [date, e.g., 20 November 2024]

   <Signed>

   [Full name, e.g., Arianti Purbasari]
   [StudentID, e.g., 05112440000xxxx]

   Also to maintain fairness for your awarded grades, state clearly in your declaration, the percentage of work and the contribution(s) for each member in your group, e.g., Arianti Purbasari 33.33% [*state Arianti's contribution(s) here*], Sri Lestari 33.33% [*state Sri's contribution(s),* and Ayu Larasati 33.33% [*state Ayu's contribution(s) here*].

5. ZIP the files of `01_linked.c`, `02_tree_from_file.c`, `03__chess_shortest_path.c`, and your declaration (e.g., `Declaration(s).PDF`) into 1 (one) only .ZIP file, namely `EF234101_FunPro(F)_Q2_StudentID1_Name1_StudentID2_Name2.ZIP`
   Send this **.ZIP** file to `yifana@gmail.com` and CC to the TA emails.

6. Have a wonderful day! Good luck! 😊