COMP 2510

Lab 9

You are expected to copy and paste your code into each corresponding box in this handout and submit it as a Word document or PDF file before the due. Additionally, your lab instructor will tell you which three questions you must showcase during the lab session. While you may demonstrate your code running in person after the due date, your file must be submitted on time.

Hint #1: For questions asking for 'a **function** which takes in xyz', the xyz refer to function parameters, not the use of scanf().

Hint #2: For questions asking for 'a **program** which reads in xyz', the xyz refer to the use of scanf().

Hint #3: For questions asking for 'a program', you are expected to include all necessary #include and define your main function. It's up to you whether to define additional helper functions, though usually, you don't need to.

Task 1: to define a **program** to allocate a memory block with the size of 10 integers, and then resize it to as big as 20 integers, deallocate it before exiting the program

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

int main(void) {
     int* ints = malloc(10 * sizeof(int));
     ints = realloc(ints, 20 * sizeof(int));

     free(ints);
     return 0;
}
```

Task 2: to define a **function** (named find_sum), to traverse a linked list to figure out and return the sum of data in all nodes

Assume: we already have the setup as below:

```
Typedef struct node node;
struct node {
        int data;
        node *next;
}
int find_sum (node* head);
```

/* return the sum of all nodes' value, return 0 if the list is empty or head is null */

```
int find_sum(node* head) {
     int sum_so_far = 0;

     node* this;
     for (this = head; this != NULL; this = this->next) {
         sum_so_far += this->data;
     }
     return sum_so_far;
}
```

Task 3: to define a **function** (named insert_asc) using what we just learnt, to traverse a linked list with double pointer to insert a value into the linked list in ascending order

Assume: we already have the setup as below:

Typedef struct node node;

struct node {

    int data;

    node *next;

}

int insert_asc(node** phead, int value) ;

/* return 1 if inserting is succeeded; return 0 otherwise */

```
int insert_asc(node** phead, int value) {
     node* new = malloc(sizeof(node));
     if (phead == NULL || new == NULL) {
         return 0;
     }
     new->data = value;

     node** curr;
     for (curr = phead; *(curr) != NULL;
          curr = &(*curr)->next) {
         if (value <= (*curr)->data) { break; }
     }

     new->next = *curr;
     *curr = new;
     return 1;
}
```

Task 4: to define a **function** (named insert_end) to insert a given value into the linked list at the end [Hints: (1) loop through the list to find the current last node, (2) may need a double pointer]

Assume: we already have the setup as below:

```
typedef struct node node;
struct node {
        int data;
        node *next;
}
```

/* return 1 if inserting is succeeded; return 0 otherwise */

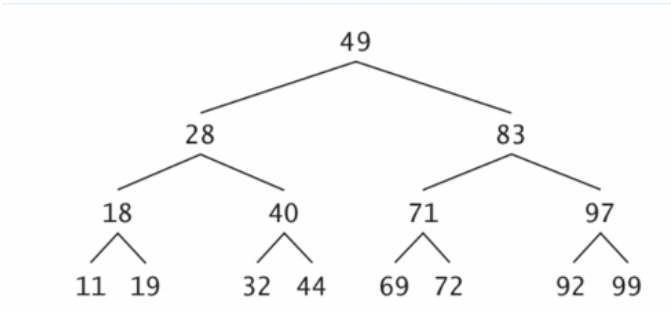
```
int insert_end(node** ptr_head, int value) {
     node* new = malloc(sizeof(node));
     if (new == NULL) {
         return 0;
     }
     new->data = value;
     new->next = NULL;
     if ((*ptr_head) == NULL) {
         *ptr_head = new;
     return 1;
     }

     node** curr;
     for (curr = ptr_head; *(curr) != NULL;
          curr = &(*curr)->next) {
         if ((*curr)->next == NULL) { break; }
     }

     (*curr)->next = new;
     return 1;
}
```

Task 5: To define a **function** (named remove) to remove all nodes with same value as the input
Assume: we already have the setup as below:
Typedef struct node node;
struct node {
    int data;
    node *next;
}
Eg, before remove(), the list as 1 -> 2 -> 3 -> 2 -> 1
    after calling remove(&head, 2); the list as 1 -> 3 -> 1

/* return 1 if removing is succeeded; return 0 otherwise */

```
int nodes_remove(node** ptr_head, int value) {
     int remove_status = 0;
     node* prev = NULL;
     node* to_remove;
     node** curr = ptr_head;
     while (*curr != NULL) {
         if (value == (*curr)->data) {
              remove_status = 0;
              if (prev != NULL) {
                   prev->next = (*curr)->next;
              }
              to_remove = (*curr);
              *curr = to_remove->next;
              free(to_remove);
              remove_status = 1;
              continue;
         } else {
              curr = &(*curr)->next;
         }
     }
     return remove_status;
}
```

Task 6: Provide the in-order, preorder and post traversals of the give binary tree



The in-order traversal is:

```
11 18 19 28 32 40 44 49 69 71 72 83 92 97 99
```

The preorder traversal is:

```
49 28 18 11 19 40 32 44 83 71 69 72 97 92 99
```

The postorder traversal is:

```
11 19 18 32 44 40 28 49 69 72 71 92 99 97 83
```

Task 7: Refer to our version of inOrder() and preOrder(), define your postOrder()
Prototype: void postOrder(TreeNode* pNode);
Same setup as:

```
typedef struct TreeNode TreeNode;
struct TreeNode {
        int data;
         TreeNode* left;
         TreeNode* right;
};
```

```
void postOrder(TreeNode* pNode) {
     if (pNode != NULL) {
         postOrder(pNode->left);
         postOrder(pNode->right);
         printf("%d ", pNode->data);
     }
}
```

Task 8: To define two programs:
The first **program** opens a file named “salutation.txt”, and then reads one line from user, writes that line into the file, close the file.

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

#define FILENAME "salutation.txt"
#define MAX_CHAR_COUNT 256

int main(void) {
     FILE* target = fopen(FILENAME, "w");
     if (target == NULL) {
         return 1;
     }
     char buffer[MAX_CHAR_COUNT] = {0};
     printf("Enter one line: ");
     fgets(buffer, MAX_CHAR_COUNT, stdin);

     fprintf(target, "%s", buffer);
     fclose(target);

     return 0;
}
```

The second **program** opens the “salutation.txt” reads the text from the file (should be one line) and then prints the text on screen(stdout).

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

#define FILENAME "salutation.txt"
#define MAX_CHAR_COUNT 256

int main(void) {
     FILE* target = fopen(FILENAME, "r");
     if (target == NULL) {
     printf("File does not exist!");
         return 1;
     }
     char buffer[MAX_CHAR_COUNT] = {0};

     fgets(buffer, MAX_CHAR_COUNT, target);
     printf("%s", buffer);

     fclose(target);

     return 0;
}
```

Task 9: To define a **program** writes the three given records into a file named "out1.txt", using fwrite function

```
typedef struct record record;
struct record{
   char name[20];
   int age;
};
```

In main(), use: record records[3] = {{"Sam", 25}, {"Tom", 30}, {"Kim", 16}};

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

#define FILENAME "out1.txt"
#define NAME_CHARS 20

typedef struct record record;
struct record {
     char name[NAME_CHARS];
     int age;
};

int main(void) {
     FILE* target = fopen(FILENAME, "w");
     if (target == NULL) {
          return 1;
     }
     record records[3] = {
          {"Sam", 25},
          {"Tom", 30},
          {"Kim", 16}
     };
     fwrite(records, sizeof(record), 3, target);
     fclose(target);
     return 0;
}
```

Task 10: Assume we have the file named "out1.txt" created by your Task9 program. Write a **program** using fwrite, fseek and fread functions changes Sam's age to 26, and then changes Kim's name to "Kimmy"

```
typedef struct record record;
struct record{
   char name[20];
   int age;
};
```

In Task 9, we had: record records[3] = {{"Sam", 25}, {"Tom", 30}, {"Kim", 16}};

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define FILENAME "out1.txt"
#define NAME_CHARS 20

typedef struct record record;
struct record {
      char name[NAME_CHARS];
      int age;
};

int main(void) {
      FILE* target = fopen(FILENAME, "r");
      if (target == NULL) {
          printf("Error opening file.\n");
          return 1;
      }
      record records[3];
      fread(records, sizeof(record), 3, target);
      fclose(target);
      for (int i = 0; i < 3; ++i) {
          if (!strcmp(records[i].name, "Kim")) {
                strcpy(records[i].name, "Kimmy");
          } else if (!strcmp(records[i].name, "Sam")) {
                records[i].age = 26;
          }
      }
      target = fopen(FILENAME, "w");
      fwrite(records, sizeof(record), 3, target);
      fclose(target);
      return 0;
}
```