CITY UNIVERSITY OF HONG KONG

Course Code & Title: EE2331 Data Structures and Algorithms

Session: Semester B, 2009-10

Time Allowed: Two hours

This paper has 7 pages (including this cover page).

Answer ALL questions in this paper.

Material, aids & instruments permitted to be used during examination:

1. Approved calculators

Section A (40%)

Question 1: There are two implementation of computing the factorial of a non-negative number.

```
int factorial_v1(int n) {
        //Precondition: n > 0
        int result = 1;
        while (n-- > 0) result *= n;
        return result;
}
```

```
int factorial_v2(int n) {
        //Precondition: n > 0
        if (n == 0) return 1;
        return (n * factorial_v2(n-1));
}
```

Which implementation is better? Please explain. **(5 marks)**

Question 2: "**unknown()**" is a recursive function. What is the output of the following program? **(5 marks)**

```
int min(int a, int b) {
        return a < b? a: b;
}

int unknown(int k, int a) {
        int i, count = 0;
        if (k <= 0) return 1;
        for (i = min(k, a); i > 0; i--)
                count += unknown(k - i, i);
        return count;
}

int main() {
        int k = 3;
        printf("%d\n", unknown(k, k));
        k = 4;
        printf("%d\n", unknown(k, k));
        k = 5;
        printf("%d\n", unknown(k, k));
        return 0;
}
```

Question 3: The inorder sequence of a binary tree is **HJKIDBCELFAG** whereas the postorder sequence is **HKDIBLFEACJG**. What is the preorder sequence of the same tree? **(5 marks)**

Question 4: The following array is the implicit array representation of a min heap tree. Show the array content after running the delete minimum operation on the tree. **(5 marks)**

[0]	[1]	...							
12	17	30	26	19	93	35	40	38	31

Question 5: State a condition that bubble sort will perform better than quicksort. Illustrate your answer with a set of sample input data. **(5 marks)**

Question 6: Quicksort will apply on the following array of integers. The pivot is selected using the median of the first, middle and last array element. Show the array content **after running the first pass** only. **(5 marks)**

[0]	[1]	...					
89	12	6	23	0	16	9	20

Question 7:

(a) Assuming that a hash table is of size 11, construct the hash table with the below data sequence while the hash function is defined as $h(i) = i \% 11$ and quadratic probing is applied,

89, 12, 6, 23, 0, 16, 9, 20

Show the contents of the hash table **after the insertion of each element**. **(6 marks)**

(b) Determine the average number of comparisons for a successful search. **(2 marks)**

(c) There is one problem for using quadratic probing. What is the problem? Explain briefly. **(2 marks)**

Section B (60%)

Question 8: (30 Marks)

The structure of the circular doubly linked list node is defined as follows:

```
struct _node {
        int data;                   //the data field of the node
        struct _node *prev;         //reference to the preceding node
        struct _node *next;         //reference to the succeeding node
}
typedef struct _node Node;
```

(a) Write a function to remove the duplicated elements in the **unordered** circular doubly linked list that has a dummy header. The function should remove all duplicated elements, but keep the first occurrence of the duplicated one in the list. **(10 Marks)**

Example:

The input list = {10, 3, 5, 5, 7, 10, 7, 5}
After removing duplicates: list = {10, 3, 5, 7}

```
void remove_duplicate(Node *head) {
        //Precondition: head points to the dummy header of a circular doubly
linked list


}
```

(b) Write a function to insert a node into an **ordered** circular doubly linked list that has a dummy header. The given linked list is already sorted in ascending order according to the values of their *data*. After the insertion, the result list should still maintain the order. You should only manipulate the pointers. Copying of *data* from one node to another is not allowed. **(6 Marks)**

Example:

The input list: {3, 5, 10}
The node *e* to be inserted to the list: 7
The result list: {3, 5, 7, 10}

```
void insert(Node *head, Node *e) {
        //Precondition: the list pointed by 'head' has dummy header


}
```

(Continued on next page)

(c) With the help of function in (a) and (b), write a function to remove the duplicated elements and perform insertion sort on a circular doubly linked list with dummy header node such that the nodes are rearranged in **ascending order** according to the values of their *data*. In your implementation, you may call the functions in (a) and (b). But copying of *data* from one node to another is now allowed.

(6 Marks)

Example:

The input list = {10, 3, 5, 5, 7, 10, 7, 5}
After sorting: list = {3, 5, 7, 10}

```
void insertion_sort(Node *head) {
        //Precondition: the list pointed by 'head' has dummy header

}
```

(d) Now you want to print out the content of the linked list in **descending order**. But you do not want to rewrite the insertion sort. Can you write a function to print out the content of a circular doubly linked list in descending order? It is given that the list has been sorted using (c) already. **(3 Marks)**

Example:

The input list = {3, 5, 7, 10}
The output = "10 7 5 3 " (The double quotations are not required)

```
void printout(Node *head) {
        //Precondition: the list has been sorted using (c) already

}
```

(e) Suppose the structure of the linked list has been changed from circular doubly to linear singly as follows:

```
struct _node {
        int data;                   //the data field of the node
        struct _node *next;         //reference to the succeeding node
}
typedef struct _node Node;
```

It is given that this list has dummy header and has been sorted in ascending order already. Can you write a <u>recursive</u> function to print out the content of the linked list in descending order? **(5 Marks)**

```
void printout_recur(Node *head) {
        //Precondition: the list has been sorted in ascending order already

}
```

Question 9: (18 Marks)

A binary tree is said to be *size-ordered* if and only if for every node the size (number of nodes) of the left subtree is greater than or equal to that of the right subtree. An empty binary tree is also *size-ordered.*

The structure of the tree node of binary tree is defined as follows:

```
struct _treenode {
        int data;                   //the data field of the tree node
        struct _treenode *left;     //reference to the left child
        struct _treenode *right;    //reference to the right child
}
typedef struct _treenode TreeNode;
```

(a) Write a recursive function to determine whether the tree pointed by *p* is *size-ordered* or not. The function should also update *size* to the number of nodes pointed by *p*.

(8 Marks)

```
int isSizeOrdered(TreeNode *p, int *size) {
        //return 1 if p is a size-ordered tree, otherwise return 0


}
```

(b) It is important to maintain a binary tree as *size-ordered* for some applications. Now write a new recursive function to change the tree to *size-ordered* if it is not *size-ordered* by flipping the appropriate left subtree with its sibling right subtree. The function should return also the size of current subtree pointed by *p*. You are NOT allowed to call the function in (a).

(10 Marks)

```
int flip(TreeNode *p) {


}
```

Question 10: (12 Marks)

A palindrome is a string that spells the same forward and backward. An example of palindrome string is "neveroddoreven".

Write a function to determine whether the input string *str* is palindrome or not using **two stacks** only. The function should return 1 if *str* a palindrome, or return 0 if *str* is not a palindrome. To simplify the function, you may assume the characters in the input string *str* are all alpha letters in small case.

The specification of the stack is defined as follows:

```
typedef struct _stack Stack;        //the name of the stack structure
void Stack_init(Stack *s);          //to initialize the stack to empty state
void Stack_destroy(Stack *s);       //to delete all data of the stack
int Stack_isEmpty(Stack *s);        //return 1 if the stack is empty,
                                    //otherwise return 0
void push(Stack *s, int e);         //to insert an element to the top of stack
int pop(Stack *s);                  //to delete and return the top element
```

Since the actual implementation of the stack is unknown to you, in your implementation of this function, you should only use the above stack operators provided to you to access the stacks. You should NOT access the internal data fields of the Stack structure directly.

```
int isPalindrome(char *str) {
        //Precondition: str is a character array that ends with '\0'
        //You can only declare at most 2 stacks and 1 character variable
        //You are NOT allowed to create other variables nor data structures


}
```

- THE END -