Spring 2021 CSE3080 Data Structures

Midterm Exam

※ Write your answers on the answer sheet. Make sure your answers are clearly recognizable.

(For all the C programs, **#include** statements are omitted due to space. Assume necessary library files are included.)

For coding problems, you should write C code in each gray box. (C++ code is allowed as well, as long as the whole program can be compiled with a C or a C++ compiler.) In each gray box, special requirements for that gray box are specified. For example, if the text in the gray box says, "Maximum number of semicolons: 3", it means in that gray box you can only write up to 3 semicolons (**;**). (a semicolon ; can be used as an end-of-statement marker or in for loop conditions.)

1. Structures - (1) 3pts, (2) 3pts, (3) 2pts - Total **8 pts**

Write code in each gray box to complete the program. The function **humans_equal** takes two arguments of type **struct Person *** and returns 1 if the two are considered the same person and return 0 if the two are considered different. In the problem, if two people have the same name, age, and salary, they are considered the same person.

Tip: **`int strcmp(const char *str1, const char* str2);`**
`Returns 0 if str1 and str2 hold equal string. Returns a non-zero value otherwise.`

(1) Define struct type here. **name** is a character array of 10 bytes, **age** is integer type, and **salary** is float type. Maximum number of semicolons: 4

```
int humans_equal(struct Person *p1, struct Person *p2) {
```

(2) Maximum number of semicolons: 4

```
}

void main() {
    struct Person p1, p2;
    strcpy(p1.name, "John"); strcpy(p2.name, "John");
    p1.age = 25; p2.age = 25;
    p1.salary = 50000; p2.salary = 50000;
```

(3) You must call **humans_equal** here. Maximum number of semicolons: 0

```
        printf("same person.\n");
    else
        printf("different people.\n");
}
```

[result]

```
same person.
```

2. Strings – (1) 2 pts, (2) 4 pts – Total **6 pts**

Write code in each gray box to complete the program. The function **mystrcat** takes two arguments of type **char*** which contain two strings and returns a **char*** which points to a concatenated string.

For this problem, you must not use any string-related library functions (functions defined in <string.h>), except **strlen**.

```
char* mystrcat(char *dst, char *src) {
  int i;
  char *rv;

  (1) Maximum number of semicolons: 1

  for(i=0; i<strlen(dst); i++) {
    rv[i] = dst[i];
  }

  (2) Maximum number of semicolons: 4

  return rv;
}

void main(int argc, char *argv[]) {
  char *c;
  char s[100] = "dog";
  char t[100] = "house";
  c = mystrcat(s, t);
  printf("%s\n", c);
}
```

[result]

```
doghouse
```

3. Stacks – (1) 2 pts, (2) 2 pts (3) 4 pts – Total **8 pts**

Write code in each gray box to complete the program. The function **push** pushes an element to the stack, and **pop** pops an element out of the stack. For this problem, you should also write what will be printed on the screen when you run this program.

```
#define MAX_STACK_SIZE 10

struct element {
    int key;
};
struct element stack[MAX_STACK_SIZE];
int top = -1;

void push(struct element item) {
    if(top >= MAX_STACK_SIZE - 1)
        stackFull();        // assume this function exists.
```

(1) Maximum number of semicolons: 1

```
}

struct element pop() {
    if(top == -1)
        stackEmpty();       // assume this function exists.
```

(2) Maximum number of semicolons: 1

```
}

void main() {
    int i;
    struct element e;

    for(i=0; i<3; i++) {
        e.key = i * 10;
        push(e);
        printf("%d inserted.\n", e.key);
    }
    printf("top: %d\n", top);

    for(i=0; i<3; i++) {
        e = pop();
        printf("%d deleted.\n", e.key);
    }
    printf("top: %d\n", top);
}
```

(3) Write what will be printed on the screen when you run this program.

4. Queues – (1) 4 pts, (2) 2 pts (3) 2 pts (4) 4 pts – Total **12 pts**

Write code in each gray box to complete the program. The function **addq** adds an element to the back of the queue, and **deleteq** removes an element from the front of the queue. The function **queueFull** checks whether space is left at the front of the array and moves elements to the front of the array to make space. For this problem, you should also write what will be printed on the screen when you run this program.

```
#define MAX_QUEUE_SIZE 10

struct element {
    int key;
};

struct element queue[MAX_QUEUE_SIZE];
int rear = -1;
int front = -1;

void queueFull() {
    int i;
    if(front == -1) {
        fprintf(stderr, "no more space in the queue\n");
        exit(1);
    }

      (1) Maximum number of semicolons: 3

    rear = rear - front - 1;
    front = -1;
}

void queueEmpty() {
    fprintf(stderr, "queue is empty.\n");
    exit(1);
}

void addq(struct element item) {
    if(rear == MAX_QUEUE_SIZE-1) queueFull();
      (2) Maximum number of semicolons: 1
}

struct element deleteq() {
    if(front == rear) queueEmpty();
      (3) Maximum number of semicolons: 1
}


void main() {
    int i;
    struct element e;

    printf("front: %2d rear: %2d\n", front, rear);
```

```
    for(i = 0; i < 8; i++) {
        e.key = i;
        addq(e);
    }

    printf("front: %2d rear: %2d\n", front, rear);

    for(i = 0; i < 8; i++) {
        e = deleteq();
    }

    printf("front: %2d rear: %2d\n", front, rear);

    for(i = 8; i < 16; i++) {
        e.key = i;
        addq(e);
    }

    printf("front: %2d rear: %2d\n", front, rear);

    for(i = 8; i < 16; i++) {
        e = deleteq();
    }

    printf("front: %2d rear: %2d\n", front, rear);
}
```

(4) Write what will be printed on the screen when you run this program.

5. Arrays – (1) 2 pts, (2) 4 pts (3) 4 pts – Total **10 pts**

Write code in each gray box to complete the program. The function **lotto** prints **n** distinct random integers from 1 to **k** on the screen. (No duplicate numbers are allowed.)

```
void lotto(int n, int k) {
    /* print n distinct random integers from 1 to k on the screen. */
    int i, j, *data;
    data = malloc(n * sizeof(int));
    data[0] = rand() % k + 1;
    for(i=1; i<n; i++) {

        (1) Maximum number of semicolons: 1

        for(j=0; j<i; j++) {

            (2) Maximum number of semicolons: 1

        }

        (3) Maximum number of semicolons: 1

    }
    for(j=0; j<n; j++) {
        printf("%d ", data[j]);
    }
    printf("\n");
    free(data);
}

void main() {
    lotto(6, 45);
}
```

[result] ※ The result may be different for each run due to randomness.

```
29 17 28 26 24 2
```

6. Linked Lists – (1) 4 pts, (2) 4 pts (3) 4 pts (4) 4 pts – Total **16 pts**

Write code in each gray box to complete the program. The function **addToList** inserts a node at the end of the list. Duplicate keys are not allowed, so if the key already exists in the list, nothing is added to the list. The function **deleteFromList** deletes a node with the given key. If the node with the key does not exist in the list, nothing is deleted. The function **printList** prints the contents of the list in the order they were inserted. The function **clearList** deletes all nodes in the list and sets the global variable **first** to NULL.

```
struct listNode {
  int key;
  struct listNode *link;
};
struct listNode *first = NULL;

void addToList(int key) {
    struct listNode *newNode;
    struct listNode *currNode, *prevNode = NULL;

    if(first == NULL) {
        newNode = malloc(sizeof(struct listNode));
        newNode->key = key;
        newNode->link = NULL;
        first = newNode;
        return;
    }

    currNode = first;
    while(1) {

        (1) Maximum number of semicolons: 3

    }
    newNode = malloc(sizeof(struct listNode));
    newNode->key = key;
    newNode->link = NULL;
    currNode->link = newNode;
}

void deleteFromList(int key) {
    struct listNode *newNode, *prevNode = NULL;
    struct listNode *currNode;

    if(first == NULL) return;
    currNode = first;

    while(currNode) {

        (2) Maximum number of semicolons: 3

    }
```

```
    if(currNode == NULL) return;

    if(prevNode != NULL) {
        prevNode->link = currNode->link;
    }

    if(currNode == first) {
        first = first->link;
    }

    free(currNode);
}

void printList() {
    struct listNode *currNode;
    currNode = first;
```

(3) Maximum number of semicolons: 3

```
}

void clearList() {
    struct listNode *temp;
```

(4) Maximum number of semicolons: 3

```
}

void main() {

    addToList(10);
    addToList(7);
    addToList(10);
    addToList(5);
    addToList(15);
    printList();

    deleteFromList(5);
    deleteFromList(10);
    deleteFromList(8);
    printList();

    clearList();
}
```

[result]

```
10 7 5 15
7 15
```

7. Binary tree traversal – (1) 4 pts, (2) 4 pts (3) 4 pts (4) 4 pts – Total **16 pts**

Write code in each gray box to complete the program. The function **recursive_inorder** prints the arithmetic expression using the infix notation, while the function **recursive_postorder** prints the arithmetic expression using the postfix notation. When you print the arithmetic expression, each expression should be printed as a single line.

For this problem, you should draw the tree structure created by the main function, and also write what will be printed on the screen when you run this program.

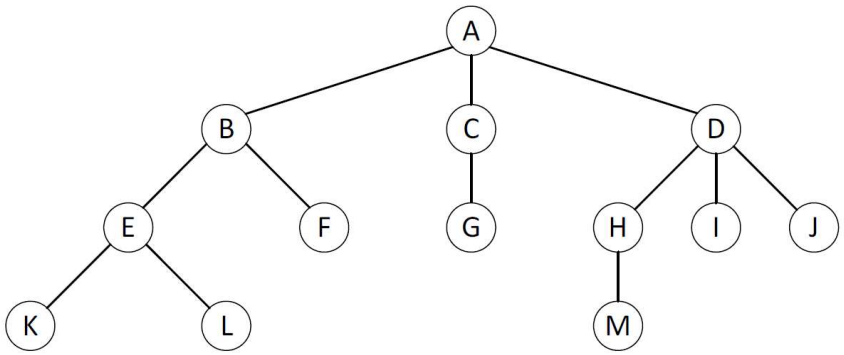
```
struct node {
  char data;
  struct node *left_child, *right_child;
};
typedef struct node *tree_pointer;

tree_pointer create_tree_node(char data) {
  tree_pointer ptr = (tree_pointer)malloc(sizeof(struct node));
  ptr->data = data;
  ptr->left_child = NULL;
  ptr->right_child = NULL;
  return ptr;
}

void recursive_inorder(tree_pointer ptr) {

    (1) Maximum number of semicolons: 3

}

void recursive_postorder(tree_pointer ptr) {

    (2) Maximum number of semicolons: 3

}

void main() {

  /* create a tree that represents an arithmetic expression */
  tree_pointer ptr, ptr1, ptr2;

  ptr1 = create_tree_node('A');
  ptr2 = create_tree_node('B');
  ptr = create_tree_node('/');
  ptr->left_child = ptr1;
  ptr->right_child = ptr2;

  ptr1 = ptr;
  ptr2 = create_tree_node('C');
  ptr = create_tree_node('*');
```

```
  ptr->left_child = ptr1;
  ptr->right_child = ptr2;

  ptr1 = ptr;
  ptr2 = create_tree_node('D');
  ptr = create_tree_node('*');
  ptr->left_child = ptr1;
  ptr->right_child = ptr2;

  ptr1 = ptr;
  ptr2 = create_tree_node('E');
  ptr = create_tree_node('+');
  ptr->left_child = ptr1;
  ptr->right_child = ptr2;

  /* call traversal functions */
  recursive_inorder(ptr);
  printf("\n");
  recursive_postorder(ptr);
  printf("\n");
}
```

(3) Draw the tree structure that is created by the main function.

(4) Write what will be printed on the screen when you run this program.

8. Tree: concepts (9 points)



(1) Consider the tree shown above. How many nodes are there in the tree? (1 pt)

(2) What is the degree of the tree? Explain why. (2 pt)

(3) Is this a binary tree? Explain why. (2 pts)

(4) What is a complete binary tree? Explain briefly. (4 pts)

9. Big-O notation (5 points)

Asymptotic time complexity is often used to evaluate algorithms. It basically indicates how fast the running time will grow with the input size. Asymptotic time complexity is expressed as a function of input size n, using the Big-O notation. Sort the following functions in the ascending order of time complexity. (The slowest growing function of n comes first.)

$O(n^2)$ $O(n^3)$ $O(\log n)$ $O(n \log n)$ $O(n)$ $O(1)$ $O(2^n)$

10. Time complexity of code blocks (10 points)

In the following problems, write the time complexity of the given code blocks using the Big-O notation, with regard to the input size **n**, which is a positive integer. **For computation cost, we will only consider how many times the statement "c = c+1;" is executed, regarding n.** (You must use **tight complexity** when using the Big-O notation.)

※ You can assume that $n = 2^m$ for some positive integer m.

(a) 2 pts

```
c = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        c = c + 1;
```

(b) 3 pts

```
c = 0;
for (i = 0; i < n*n; i++)
    for (j = 0; j < i; j++)
        c = c + 1;
```

(c) 2 pts

```
c = 0;
for (i = 1; i <= n; i = i + 1)
    for (j = 1; j <= n; j = j * 2)
          c = c + 1;
```

(d) 3 pts

```
j = 1; c = 0;
for(i = 1; i <= n; i = i + 1) {
    for(k = 1; k <= j; k = k + 1) {
        c = c + 1;
    }
    j = j * 2;
}
```

- End of the exam -