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.)

***** 1~7 코드 문제 기준 *****

- 괄호, 세미콜론, 철자 문제 있는 경우 -1점
- 변수명 잘못된 경우(철자 실수로 볼 수 없는 경우) 해당 문제 0점
- 세미콜론 개수를 초과한 경우 0점 (단, for 세미콜론을 제외하고 작성한 경우 추가된 세미콜론당 -1점)
- 선언 없이 새로운 변수를 사용한 경우 해당 라인 인정 X
- 모든 코드는 답과 관계없이 의도에 맞게 구현된 경우 정답으로 인정

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)

```
struct Person {
  char name[10];
  int age;
  float salary;
};
```

- `struct`가 제대로 선언되지 않은 경우 -2점 (typdedef)
- 그 외 선언이 잘못된 경우 각각 -1점

(2)

```
if (strcmp(p1->name, p2->name)) return 0;
if (p1->age != p2->age) return 0;
if (p1->salary != p2->salary) return 0;
return 1;
```

- if 조건당 1점 (단, if 안에 여러 조건으로 이루어진 경우 모두 정확해야 함)
- return 값이 반대로 되거나 1,0이 아닌 다른 값인 경우 -2점 (단, return 값이 TRUE,FALSE 인 경우는 0점)
- 모든 조건에 대해 return 값이 없는 경우 -1점
- 포인터 접근이 잘못된 경우 -2점

(3)

```
if (humans_equal(&p1, &p2))
```

- return 값이 반대로 되거나 1,0이 아닌 다른 값인 경우 1점 (1(2)과 연결되어야 함)
- 주소값이 잘못된 경우 0점
- 조건이 잘못된 경우 0점

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**.

(1)

```
rv = (char*)malloc(strlen(dst)+strlen(src)+1);
rv = malloc(strlen(dst)+strlen(src)+1);
rv = calloc(strlen(dst)+strlen(src)+1, sizeof(char));
```

- +1을 안한 경우 0점
- 사이즈를 199이상으로 한 경우 1점
- sizeof(char)이 아닌 다른 타입을 곱한 경우 -1점

(2)

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

- for문 2점 + 마지막 ₩0 처리 2점
- for문 인덱스 잘못된 경우 인정 X
- ₩0 인덱스가 잘못된 경우 -1점
- ₩0이 아닌 NULL은 인정 X
- 그 외 문제가 있는 경우 0점

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.

(1)

```
stack[++top] = item;
stack[++top].key = item.key;
```

- ++top 1 점
- top++는 0 점
- 한쪽만 .key 인 경우 0 점
- 변수명 틀린 경우 0 점
- else {} 인정

(2)

```
return stack[top--];
```

- top-- 1 점
- --top 는 0 점
- 변수명 틀린 경우 0 점
- else {} 인정

(3)

```
0 inserted.
10 inserted.
20 inserted.
top: 2
20 deleted.
10 deleted.
0 deleted.
top: -1
```

- top 2 점 + inserted 1 점 + deleted 1 점
- format 적용 안한 경우 top/inserted/deleted 각각 -1 점
- top 도 함께 iterate 에 포함시킨 경우 -1 점

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.

(1)

```
for (i=front+1; i<MAX_QUEUE_SIZE; i++) {
  queue[i-(front+1)] = queue[i];
}
```

```
for (i=front+1; i<=rear; i++) {
  queue[i-(front+1)] = queue[i];
}
```

```
for (i=front; i<rear; i++) {
  queue[i-front] = queue[i+1];
}
```

```
for (i=0; i<rear-front; i++) { // i<rear-front+1 조건은 -1점
  queue[i] = queue[i+front+1];
}
```

- 위의 코드에서 인덱스를 동일하게 움직이는 경우 모두 인정
- queue[].key = queue[].key는 인정
- else {} 안에 코드를 적은 경우 -1점
- index 이동이 잘못된 경우 0점

(2)

```
queue[++rear] = item;
```

```
queue[++rear].key = item.key;
```

- ++rear 1 점
- rear++는 0 점
- 한쪽만 .key 인 경우 0 점
- else {} 안에 적은 경우 인정

(3)

```
return queue[++front];
```

- ++front 1 점
- front++는 0 점
- 한쪽만 .key 인 경우 0 점
- else {} 안에 적은 경우 인정

(4)

```
front: -1 rear: -1
front: -1 rear: 7
front: 7 rear: 7
front: -1 rear: 7
front: 7 rear: 7
```

- front 2점 + rear 2점
- 중간에 no more ~, queue is ~ 등 적은 경우 등장마다 -1점
- 01, 07과 같이 적은 경우 -1점

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.)

(1)

```
data[i] = rand()%k+1;
```

- data[i]가 아닌 다른 변수에 rand()%k+1 을 정확하게 저장한 경우 1 점
  (이때, i,k 와 같은 의미 있는 변수 또는 선언 안한 새로운 변수는 인정 X)
- index 가 잘못된 경우 0 점
- rand()가 아닌 rand 라고 적은 경우 0 점

(2)

```
if (data[j] == data[i]) break;
```

- 부분 점수 x

(3)

```
if (j < i) i--;
if (j != i) i--;
```

- 부분 점수 x

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.

(1)

```
if (currNode->key == key) return;
if (currNode->link == NULL) break;
currNode = currNode->link;
```

- 무한루프 종료 조건 없는 경우 0 점
- key 비교문 1 점 + currNode 관련 3 점
- currNode 관련 순서가 정확해야 함.
- key 비교가 뒤에 오는 경우 -2 점
- 포인터 잘못 쓴 경우 -2 점

(2)

```
if (currNode->key == key) break;
prevNode = currNode;
currNode = currNode->link;
```

- if 조건문 2 점 + prevNode 1 점 + currNode 1 점
- if 와 link 이동 순서가 잘못된 경우 1 점
- 포인터 잘못 쓴 경우 -2 점

(3)

```
while (currNode) {
  printf("%d ", currNode->key);
  currNode = currNode->link;
}
printf("\n");
```

- 1 줄당 1 점
- while(currNode->link) 1 점
- 무한루프 등 while 문 제대로 되지 않는 경우 0 점
- 포인터 잘못 쓴 경우 -2 점

(4)

```
while (first) {
  temp = first;
  first = first->link;
  free(temp);
}
```

```
while (first) {
  temp = first->link;
  free(first);
  first = temp;
}
```

- free() 대신 deleteFromList()이용한 경우 인정
- free 가 제대로 이루어지지 않는 경우 0 점
- 포인터 잘못 쓴 경우 -2 점

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.

(1)

```
if (ptr) {
  recursive_inorder(ptr->left_child);
  printf("%c ", ptr->data);
  recursive_inorder(ptr->right_child);
}
```

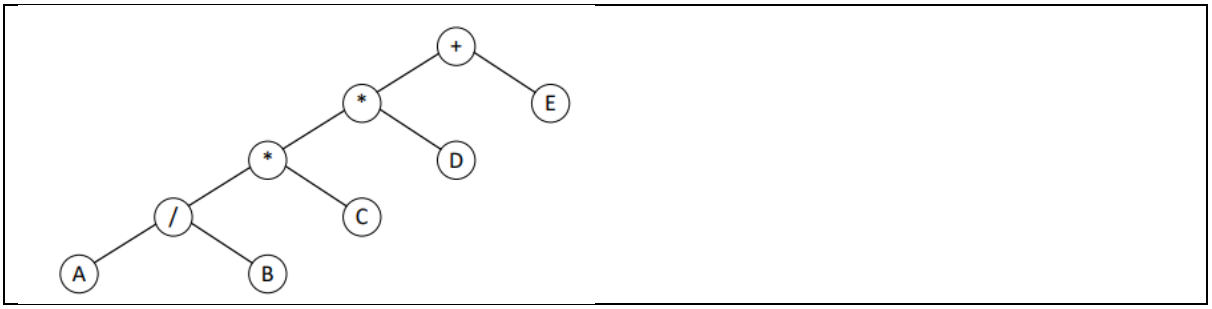
- inorder 순서 및 문법을 정확하게 적은 경우 앞의 조건과 관계없이 2 점

(2)

```
if (ptr) {
  recursive_postorder(ptr->left_child);
  recursive_postorder(ptr->right_child);
  printf("%c ", ptr->data);
}
```

- postorder 순서 및 문법을 정확하게 적은 경우 앞의 조건과 관계없이 2 점

(3)



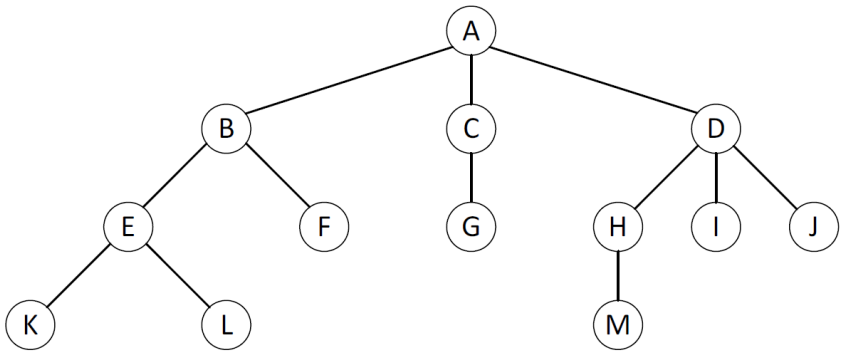
- 부분 점수 x

(4)

```
A / B * C * D + E
A B / C * D * E +
```

- 각각 2 점

8. Tree: concepts (9 points)



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

13

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

3. The maximum number of children of a node is 3.

- 답 1 점 + 이유 1 점

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

No. In a binary tree all nodes should have less than or equal to 2 children.

- 답 1 점 + 이유 1 점

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

A complete binary tree is a binary tree where all the levels are completely filled except possibly the last level and the last level has all the keys as left as possible.

- 한 level 을 다 채운 후 다음 level 로 넘어감 2 점 + level 에서 왼쪽부터 채움 2 점

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.)

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

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

- 아무 명시없이 반대로 쓴 경우 -1 점
- O()를 쓰지 않은 경우 -2 점
- 하나 잘못될 때마다 -1 점

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 $O(n^2)$

(b) 3 pts $O(n^4)$

(c) 2 pts $O(nlogn)$

(d) 3 pts $O(2^n)$

- 부분 점수 x