Fall 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, "max 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.)

For other problems, just write your answers according to the instructions.

1. (1) 5 pts – Total **5 pts**

The function **print1** prints the memory address of each element along with their values. We assume that the size of data type **int** is 4 bytes.

```
void print1(int *ptr, int rows) {
    int i;
    for(i = 0; i < rows; i=i+2)
        printf("%p    %d\n", ptr+i, *(ptr+i));
    printf("\n");
}

void main() {
    int one[] = {0, 1, 2, 3, 4};
    print1(one, 5);
}
```

(1) Write what will be printed on the screen when you run this program. (The first line is given.)

```
0x7ffd997866a0    0
```

2. (1) 5 pts – Total **5 pts**

Write code in each gray box to complete the program. The function **sum** takes two arguments. The first argument is the pointer to an array of integers, and the second argument is the number of elements in the array. The function returns the sum of the integers in the array.

```
#define MAX_SIZE 100

int sum(int *list, int n) {
    int i;

   (1) max number of semicolons: 4

    return total;
}

int main() {
    int input[MAX_SIZE], answer;
    int i;
    for(i=0; i<MAX_SIZE; i++)
        input[i] = i;
    answer = sum(input, MAX_SIZE);
    printf("The sum is: %d\n", answer);
    return 0;
}
```

[result]

```
4950
```

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

Write code in each gray box to complete the program. The function **make2dArray** takes two integer arguments, rows and cols, and dynamically allocates memory to the 2d array. Then it returns the pointer to the 2d array.

In the code, you must NOT include numeric literals (숫자).

Tip: `void *malloc(size_t size)` allocates the requested memory and returns a pointer to it. `size` is in bytes.

```
int** make2dArray(int rows, int cols) {

    int **x, i;

    (1) allocate memory for row pointers; max number of semicolons: 1

    for(i = 0; i < rows; i++)  {
        (2) allocate memory for each row; max number of semicolons: 1
    }

    return x;
}

void main() {
    int i, j;
    int **matrix;

    matrix = make2dArray(5, 5);

    for(i = 0; i < 5; i++) {
        for(j = 0; j < 5; j++) {
            matrix[i][j] = i*5 + j + 1;
            printf("%2d " matrix[i][j]);
        }
        printf("\n");
    }
}
```

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

4. (1) 5 pts – Total **5 pts**

Answer the question according to the code below.

```
typedef struct {
  enum tagField {pitcher, hitter} role;
  union {
    int SO;
    int HR;
  } u;
} playerType;

typedef struct {
  char name[40];
  int age;
  playerType playerInfo;
} baseballPlayer;

void printPlayerInfo(baseballPlayer p) {

    printf("name: %s\n", p.name);
    printf("age: %d\n", p.age);
    if(p.playerInfo.role == hitter)
      printf("HR: %d\n", p.playerInfo.u.HR);
    else
      printf("SO: %d\n", p.playerInfo.u.SO);
}

void main() {

  baseballPlayer person1, person2;

  strcpy(person1.name, "Hyun-Jin Ryu");
  person1.age = 32;
  person1.playerInfo.role = pitcher;
  person1.playerInfo.u.SO = 163;
  person1.playerInfo.u.HR = 3;

  strcpy(person2.name, "Shin-Soo Choo");
  person2.age = 37;
  person2.playerInfo.role = hitter;
  person2.playerInfo.u.HR = 24;
  person2.playerInfo.u.SO = 170;

  printPlayerInfo(person1);
  printPlayerInfo(person2);
}
```

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

5. (1) 6 pts, (2) 4 pts – Total **10 pts**

Write code in each gray box to complete the program. The function **transpose** transposes (전치) the matrix (switches the row and column indices of the matrix). Note that the 2d array **m** is declared as a global variable.

```
int m[6][6];

void printMatrix(int rows, int cols) {
  int i, j;
  for(i=0; i<rows; i++) {
    for(j=0; j<cols; j++)
      printf("%3d ", m[i][j]);
    printf("\n");
  }
  printf("\n");
}

void transpose(int rows, int cols) {
  int i, j, k;
  for(i=0; i<rows; i++) {
    for(j=i+1; j<cols; j++) {
        (1) max # of semicolons: 3
    }
  }
}

void main() {
  int i, j;
  for(i=0; i<6; i++) {
    for(j=0; j<6; j++) {
      m[i][j] = 0;
    }
  }

  m[0][0] = 15;
  m[0][3] = 22;
  m[0][5] = -15;
  m[1][1] = 11;
  m[1][2] = 3;
  m[2][3] = -6;
  m[4][0] = 91;
  m[5][2] = 28;

  transpose(6, 6);
  printMatrix(6, 6);
}
```

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

6. (1) 10 pts – Total **10 pts**

Answer the question according to the code below.

Tip: `char* strcat(char* destination, const char* source);`

The `strcat()` function appends a copy of the null-terminated string pointed by the source to the null-terminated string pointed to the destination. The first character of the source overwrites the null-terminator of destination.

```
void main() {
  char s[] = "dog";
  char t[] = "house";
  char u[] = "chocolate";
  printf("%p: %s\n", s, s);
  printf("%p: %s\n", t, t);
  printf("%p: %s\n", u, u);
  strcat(s, u);
  printf("%s\n", s);
  printf("%s\n", t);
  printf("%s\n", u);
}
```

(1) Write what will be printed on the screen when you run this program. The first three lines are given. Write the remaining lines.

```
0x7ffce8fcf104: dog
0x7ffce8fcf108: house
0x7ffce8fcf10e: chocolate
```

7. (1) 3 pts, (2) 4 pts, (3) 4 pts – Total **11 pts**

Write code in each gray box to complete the program. The function **strnins** takes three arguments: two strings **s** and **t**, and one integer **pos**. The function creates a new string where **t** is inserted in the middle of **s**, at the position indicated by **pos**. For example, if **s** is "amobile", **t** is "uto", and **pos** is 1, the function returns string "automobile".

For this problem, you must not use any string-related library functions (functions defined in <string.h>), except **strlen**. The `strlen()` function returns the length of a given string.

```
char* strnins(char* s, char* t, int pos) {
    char *str;
    int i;

    if(pos < 0 || pos > strlen(s)) {
        fprintf(stderr, "pos is out of bounds.\n");
        exit(1);
    }

    str = (char*)malloc(strlen(s) + strlen(t) + 1);

    for(i=0; i<pos; i++) {
        (1) max # of semicolons: 1
    }
    printf("%s\n", str);
    for(i=0; i<strlen(t); i++) {
        (2) max # of semicolons: 1
    }
    printf("%s\n", str);
    for(i=pos; i<strlen(s); i++) {
        (3) max # of semicolons: 1
    }
    return str;
}

void main() {
    char s[] = "amobile";
    char t[] = "uto";
    char *str = strnins(s, t, 1);
    printf("%s\n", str);
}
```

[result]

```
a
auto
automobile
```

8. (1) 4 pts, (2) 4 pts (3) 4 pts – Total **12 pts**

Write code in each gray box to complete the program. This program implements a stack using a linked list. The elements of the stack are integers. The function **push** pushes an element to the stack, and **pop** pops an element out of the stack.

```
#define IS_FULL(x) !x
#define IS_EMPTY(x) !x
struct stack {
  int item;
  struct stack* link;
};
typedef struct stack* stack_pointer;
stack_pointer top;

void push(stack_pointer *top, int item) {
  stack_pointer temp = (stack_pointer)malloc(sizeof(struct stack));
  if(IS_FULL(temp)) {
    fprintf(stderr, "the memory is full\n");
    exit(1);
  }
```

(1) max # of semicolons: 3

```
}

int pop(stack_pointer *top) {
  stack_pointer temp = *top;
  int item;
  if(IS_EMPTY(temp)) {
    fprintf(stderr, "the stack is empty\n");
    exit(1);
  }
```

(2) max # of semicolons: 2

```
  free(temp);
  return item;
}

void printStack(stack_pointer top) {
  stack_pointer temp = top;
  while(temp != NULL) {
    printf("%2d ", temp->item);
    temp = temp->link;
  }
  printf("\n");
}
```

```
void main() {
  int i;
  for(i=0; i<5; i++) {
    push(&top, i);
    printStack(top);
  }
  for(i=0; i<5; i++) {
    pop(&top);
    printStack(top);
  }
}
```

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

9. (1) 4 pts, (2) 4 pts (3) 4 pts – Total **12 pts**

Write code in each gray box to complete the program. The function **addq** adds an element to the queue, and **deleteq** deletes an element out of the queue. For this problem, you should also write what will be printed on the screen when you run this program.

```
#define IS_FULL(x) !x
#define IS_EMPTY(x) !x

struct queue {
  int item;
  struct queue *link;
};
typedef struct queue *queue_pointer;
queue_pointer front, rear;

void addq(queue_pointer *front, queue_pointer *rear, int item) {
  queue_pointer temp = (queue_pointer)malloc(sizeof(struct queue));
  if(IS_FULL(temp)) {
    fprintf(stderr, "the memory is full\n");
    exit(1);
  }
  temp->item = item;
  temp->link = NULL;
```

(1) max # of semicolons: 3

```
}

int deleteq(queue_pointer *front) {
  queue_pointer temp = *front;
  int item;
  if(IS_EMPTY(*front)) {
    fprintf(stderr, "the queue is empty\n");
    exit(1);
  }
```

(2) max # of semicolons: 2

```
  free(temp);
  return item;
}

void printQueue(queue_pointer front) {
  queue_pointer temp = front;
  while(temp != NULL) {
    printf("%2d ", temp->item);
    temp = temp->link;
  }
  printf("\n");
}
```

```
void main() {
  int i;
  for(i=0; i<5; i++) {
    addq(&front, &rear, i);
    printQueue(front);
  }

  for(i=0; i<5; i++) {
    deleteq(&front);
    printQueue(front);
  }
}
```

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

10. Big-O notation – Total **5 pts**

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.)

$O(n^2)$ $O(\sqrt{n})$ $O(n \log n)$ $O(1)$ $O(\log n)$ $O(2^n)$ $O(n)$

11. Time complexity of code blocks – Total **15 pts**

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.

(1) 3 pts

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

(2) 3 pts

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

(3) 3 pts

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

(4) 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;
}
```

(5) 3 pts

```
i = 0; j = 0; c = 0;
while (j <= n) {
    i = i + 1;
    j = j + i;
    c = c + 1;
}
```

- End of the exam -