C Programming Language: mid-term exam (180 minutes)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Total
5	5	10	10	10	15	10	5	10	10	10	10	10	5	10	135

1. [5] Describe how to write and compile a simple program that prints out a sentence "Hello C" using editor vi and gcc compiler.

2. **[5]** Do we always need to include *stdio.h*? Explain.

3. **[10]** Is the following program segment legal in C? If legal, what is the result? Explain.
   (a) int x=0; if (x = 1) printf("Yes\n"); else printf("No\n");
   (b) x = (3<2) ? 3 : 2>1 ? 2 : 1; printf("%d\n", x);
   (c) x = 1; printf("%d\n", x++); printf("%d\n", ++x);
   (d) for(x=2;x=1;x=0) printf("%d\n", x);
   (e) i = 0; while (i <10){ i++; break; } printf("%d\n", i);

4. **[10]** What is the output of the following codes? Make your assumption and Explain your answers.
   int x[2]={1, 8}, *p=&x[0];

   p=&x[0];x[0]=1; x[1]=8; x[0] = *p++ + 1; printf("x[0]=%d, x[1]=%d, *p=%d, p=%p\n", x[0], x[1], *p, p);
   p=&x[0];x[0]=1; x[1]=8; x[0] = (*p)++ + 1; printf("x[0]=%d, x[1]=%d, *p=%d, p=%p\n", x[0], x[1], *p, p);
   p=&x[0];x[0]=1; x[1]=8; x[0] = *++p + 1; printf("x[0]=%d, x[1]=%d, *p=%d, p=%p\n", x[0], x[1], *p, p);
   p=&x[0];x[0]=1; x[1]=8; x[0] = ++*p + 1; printf("x[0]=%d, x[1]=%d, *p=%d, p=%p\n", x[0], x[1], *p, p);

5. [10] Write a program compiled as *poly* to compute a polynomial of $A_nx^n + A_{n-1}x^{n-1} + A_{n-2}x^{n-2} + \ldots + A_2x^2 + A_1x^1 + A_0$. So by using command line, we can call 'poly 4 3 2 -5 -1 7 -6' as an example to compute $3x^5 + 2x^4 - 5x^3 - x^2 + 7x - 6$ *for* $x=4$. In the program, you have to write a function int poly(int coef[], int n, int x) to compute the polynomial. Note that you can use atoi(string) to convert a string to an integer. For example, k = atoi("10") assigns 10 to variable k.

6. [15] Explain what is "Call-by-value"? Why do we need the feature of "Call-by-value" to make the recursion work? Please think how it is related to stack memory and use the recursive function that computes Fibonocci numbers (*int Fibonocci(int n)*) as an example. Could you write a new recursive version of *int Fibonocci(void)* function that does not have any parameter but use a global variable *n* and local variables in *Fibonocci(void)* to compute f_n. Note that *Fibonocci* numbers are defined as follows: $f_0 = 0$, $f_1 = 1$, and $f_n = f_{n-1} + f_{n-2}$.
   Also, write three versions of function to compute **f** defined as: $f_0 = 0$, $f_1 = 1$, $f_2 = 2$, and $f_i = f_{i-1} + f_{i-2} + f_{i-3}$, for $i \geq 3$,
   (1) purely recursive
   (2) iterative
   (3) a modifiend recursive function using recursion and an array to save time.

7. [10] Write a program that prints an nxn magic square (a square arrangement of the numbers 1, 2, …, n^2 in which the sums of the rows, columns, and diagonals are all the same). The users will specify the value n and program outputs should be as follows.
   This program creates a magic square of a specified size.
   The size must be an odd number between 1 and 99.
   Enter the size of magic square: 5

17	***24***	***1***	***8***	***15***
23	***5***	***7***	***14***	***16***
4	***6***	***13***	***20***	***22***
10	***12***	***19***	***21***	***3***
11	***18***	***25***	***2***	***9***

   Start by placing the number 1 in the middle of row 0. Place numbers 2, 3, …, n^2 sequentially into the next cell by moving up one row and right one column. Any attempt to go outside the bounds of the square should "wrap around" to the opposite side of the square. If we are trying find a cell for number *k* and the cell for *k* is already occupied, we put number *k* directly below the cell occupied by number *k* – 1.

8. [5] Given the following declarations,
   int x=10;
   int *y = &x;
   int **z = &y;
   printf("%d, %d, %d, %d, %d, %d\n", x, y, *y, z, *z, **z);
   What got printed? (Make any assumption if you need and show your answer by drawing a graph)

9. [10] Write your own version of the function void compute_AB(int x[], int y[], int n, int *numA, int *numB) to compute how many A's and B's for master mind program, where array x[] stores the answers of the color numbers and y[] stores the guesses and we assume the **color numbers can be repeated**. Note that we assume the valid numbers are 0 to n – 1. Please complete the program based on the following pseudo code.

```
main()
{
   //Enter n;
   //while(1){
   //    input guesses
   //    compute_AB();
   //    output_AB();
   //    check if correct and break the loop
   //}
}
```

Given the following function, please trace it and check if it works or not. Why?

```
void compute_AB(int x[], int y[], int n, int *numA, int *numB)
{
   int i, j;
   *numA = *numB = 0;
   for (i=0; i<n; i++) if(x[i]==y[i]) *numA++;
   for (i=0; i<n; i++){
        if(x[i]==y[i]) continue;
        for (j=0; j<n; j++){
              if(y[i]==x[j]) {*numB++; break;}
        }
   }
}
```

Given the following function, please trace it and check if it works or not. Why?

```
void compute_AB(int x[], int y[], int n, int *numA, int *numB)
{
   int i, j;
   *numA = *numB = 0;
   for (i=0; i<n; i++) if(x[i]==y[i]) {*numA++; x[i]=-1; y[i]=-2;}
   for (i=0; i<n; i++)
        for (j=0; j<n; j++) if(y[i]==x[j]) {*numB++; x[j]=-1; break;}
}
```

10. [10] Consider the 7-segment LEDs. To form a digit we turn on some of the seven segments while leaving others off. Suppose that we want to set up an array that remembers which segments should be on for each digit by numbering them as shown in Figure 1. We can declare the digits 0 to 9 as follows: cons tint segments[10][7] = {{1,1,1,1,1,1,0}, …};
    (a) Please complete the above initializers.
    (b) Write a complete program that simulates a 24 hour clock (2 digits for hours and 2 digits for minutes) as shown in Figure 2 by using function sleep(1) to delay one second in each loop of the while(1) infinite loop to simulate one minute, like the following program. Note that the vertical and horizontal bars are represented by a vertical sign and two dashes in the text mode, respectively. Please try to use loops as many as possible to write the function display().

```
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
// function prototypes are here
main()
{
   const int segments[10][7] = {{1,1,1,1,1,1,0}, …};
   int digit[4]={0,0,0,0};
   …
   while(1){
      printf("0123456789012345678 9\n");
      display(digit);
      sleep(1);
      digit_compute(digit);
   }
   return 0;
```

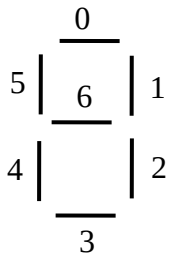


Figure 1

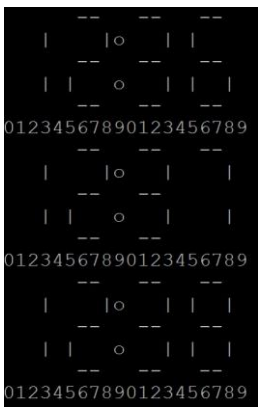


Figure 2

}

11. [10] A random number generator is written as follows. Let's say we have a program ***PRG*** that calls random() in (a) 10 times. We know that we will get exactly the same sequence of 10 numbers every time we run the program ***PRG***. Thus, we should have another function called ***set_seed()*** to set the variable ***seed*** to different value every time the program PRG starts. Based on this version of random number generator, is it possible to write the function ***set_seed()*** that runs correctly? If not, how to change the following code so that we can write a correct ***set_seed()*** function? Please describe your design of the function ***set_seed().*** Also, please write another function to compute how many different numbers this random number generator can generate?

```
#define INITIAL_SEED 17
#define MULTIPLIER 25173
#define INCREMENT 13849
#define MODULUS 65536
unsigned random(void){
    static unsigned seed=INITIAL_SEED;
    seed = (MULTIPLIER * seed + INCREMENT) % MODULUS;
    return seed;
}
```

12. [10] Please write a complete program called *max* that can take any number of input values by command line and out the maximum. For example, run "max 1 2 3 4" will output 4 and run "max 3 2 1 4 5 8 9" will output 9.

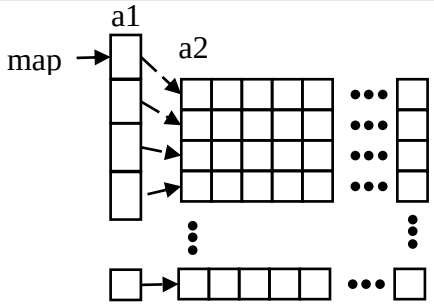
13. [10] Write a recursive program ***combination***(A, n, k) that you can print out all the combinations of *k* numbers out of *n* numbers stored in an array A with additional rule that **the sequence of these *k* numbers must be in an increasing order**. For example, if there are 4 numbers (4, 1, 2, 3) stored in an array A[4], calling this recursive function *combination*(A, 4, 2) can get a result of (1, 2), (1, 3), and (2, 3) (no (4, 1), (4, 2), (4, 3),), or calling *combination*(A, 4, 3) can get a result of (1, 2, 3), (no (4, 1, 2), (4, 1, 3) and (4, 2, 3)). Your recursive program must consider to avoid the unnecessary recursive function calls. Don't first generate all the combinations and then check if each combination is in an increasing order.

14. [5] The maximum and minimum int values are INT_MAX=2147483647(2^{31}-1) and INT_MIN = -2147483648(-2^{31}), respectively. Which header file defines these two constants? Write a function to check if the product of two integers is between -2147483648 and 2147483647 inclusively and no overflow occurs.

15. [10] Consider the ***mine*** program (踩地雷遊戲) listed in the last page. The variables *map*, *a1*, *a2*, *mine_cnt*, and *n* are global (*int **map, *a1[N], a2[N][N], mine_cnt, n = N*;).
    (a) Trace the program and figure out what are the outputs of the program.
    (b) The function initialization() is too long. Please rewrite the program such that no global variable is used and the code segments part 1 and 2 are isolated as two individual functions: *generate_mine_positions()* and *compute_mines()*.

```c
#include <stdio.h>
#include <stdlib.h>
#define N 100
int **map, *a1[N], a2[N][N], n = N;
int mine_cnt;
void display(int **map, int nx, int ny, int method);
void initialization(int argc, char *argv[]);
int game_over(int **map, int nx, int ny);
int main(int argc, char *argv[])
{
   int i= 10, j, x, y;
   initialization(argc, argv);
   while(1){
        system("clear");
        display((int **)map, n, n, 0);
        scanf("%d%d", &x, &y);
        if ( x<0 || x >n-1|| y> n-1|| y < 0) break;
        if(map[x][y] == 9){
           printf("you are dead!\n GAME OVER \n");
           display((int **)map, n, n, 1); break;
        }
        map[x][y] += 10;
        if(game_over(map, n, n) == 1){
             printf("you win!!! \n GAME OVER \n");
             display((int **)map, n, n, 1);break;
        }
   }
} /* end main */
```

```c
void display(int **map, int nx, int ny, int method)
{
 int i, j;
 printf("      ");
 for (i=0; i< nx; i++) printf("%d ", i%10);
 printf("\n\n");
 for (i=0; i< nx; i++){
    printf("%d    ", i%10);
    for (j=0; j< ny; j++){
       switch(method){
       case 0: if(map[i][j] == 10) printf("- ");
                 else if(map[i][j] < 10) printf("0 ");
                 else printf("%d ", map[i][j]%10);
                 break;
               default:
                    if(map[i][j] == 9) printf("X ");
                    else printf("%d ", map[i][j]%10);
       }
    }
   printf("\n");
 }
}
```




```c
void initialization(int argc, char *argv[])
{
   int i, j, l, m, x, y, cnt;
   if(argc != 3){
      printf("please run as \"%s n m\" \n", argv[0]);
      printf("n = size & m = # of mines \n");
      exit(1);
   }
   n = atoi(argv[1]);
   mine_cnt = atoi(argv[2]);
   map = (int **) a1;
   for (i=0; i< n; i++) map[i] = (int *)a2[i];

   for (i=0; i< n; i++)
        for (j=0; j< n; j++)
      map[i][j] = 0;
   srand(time(NULL));
   cnt = 0;
   while(1){
        x = rand()%n;
        y = rand()%n;

        printf("%d %d \n", x, y);
        if (map[x][y]== 9) continue; /* 9 = mine */
        map[x][y] = 9;
        if(++cnt == mine_cnt) break;;
   }
   for (i=0; i< n; i++)
        for (j=0; j< n; j++){
      if (map[i][j]==9) continue;
      for(l = i-1; l <= i+1; l++){
                  if( l<0 || l >= n) continue;
            for(m=j-1; m<=j+1; m++){
                  if( m<0 || m >= n) continue;
                  if( l==i && m == j) continue;
                  if (map[l][m]==9) map[i][j]++;
            }
      }
        }
}
```

```c
int game_over(int **map, int nx, int ny)
{
   int i, j;
   for (i=0; i< nx; i++){
        for (j=0; j< ny; j++){
      if(map[i][j] < 9) return 0;
        }
   }
   return 1;
}
```