2023 C Programming Language: Midterm exam

1	2	3	4	5	6	7	8	9	10	11	12	13	14	Total
5	10	10	5	5	5	10	10	10	10	10	10	10	10	120

1. [5] Is the following program segment legal in C? If legal, what is the result. Explain.
   (a) int x=0; if (x = 0 || x = 1) printf(“%d\n”, x);
   (b) int x=0; if (x = (3>2) ? 3 : 2+1 ? 2 : 1) printf("%d\n", x);
   (c) while(1){ int x=1;
   printf(“%d\n”, x);
   {
   int x=2, y=3;
   printf(“%d %d \n”, x, y);
   }
   if (printf(“%d %d \n”, x, y) > 3) break;
   }
   (d) int x=0; for (x=0;x=0;x=0) printf(“%d\n”, x);
   (e) int x=0;
   while (x <10){ x++; break; printf(“%d\n”, x); }

Q1-a: X
cc.c:14:18: error: lvalue required as left operand of assignment
14 | if (x = 0 || x = 1) {
| ^
Explain:

```
if (x = (0 || x) = 1) {
   printf("%d\n", x);
 }
```

Q1-b: O
3
Q1-c:
error: 'y' undeclared (first use in this function)
19 | if (printf("%d %d \n", x, y) > 3) break;
| ^
cc.c:19:28: note: each undeclared identifier is reported only once for each function it appears in

Q1-d: O
空白

```
PS C:\Users\louis\Desktop> cd "c:\Users\louis\Desktop\" ; if ($?) { gcc cc.c -o cc }
; if ($?) { .\cc }
PS C:\Users\louis\Desktop>
```

Q1-e: O
空白

```
PS C:\Users\louis\Desktop> cd "c:\Users\louis\Desktop\" ; if ($?) { gcc cc.c -o cc }
; if ($?) { .\cc }
```

2. [10] Use the random number generator to write a complete program that can generate *n* random numbers ranged from *min* to *max* by running "***myrandom n min max***" in command line. Also answer what is the maximum number that function rand() can generate and where it is defined? Why we have to set seed, usually using time()?

Q2:
PS C:\Users\louis\Desktop> gcc -o cc cc.c
PS C:\Users\louis\Desktop> ./cc 5 10 100
48 58 13 100 72

```
PS C:\Users\louis\Desktop> gcc -o cc cc.c
PS C:\Users\louis\Desktop> ./cc 5 10 100
48 58 13 100 72
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char* argv[]) {
  // Check if the correct number of command-line arguments are provided
  if (argc != 4) {
    printf("Usage: %s n min max\n", argv[0]);
    return 1;  // Exit with an error code
  }
  // Parse command-line arguments
  int n = atoi(argv[1]);
  int min = atoi(argv[2]);
  int max = atoi(argv[3]);
  // Validate input range
  if (min > max) {
    printf("Error: min should be less than or equal to max.\n");
    return 1;  // Exit with an error code
  }
  // Set seed using current time
  srand((unsigned)time(NULL));
  // Generate and print n random numbers
  for (int i = 0; i < n; ++i) {
    int randomNumber = rand() % (max - min + 1) + min;
    printf("%d ", randomNumber);
  }
```

```
  printf("\n");
  return 0;
}

// According to the C standard (specifically, the C99
standard and later), the
// minimum and maximum values for the return value of rand()
are defined by the
// constants RAND_MAX(32767) and 0
// #include <stdio.h>
// #include <stdlib.h>

// int main() {
//     printf("RAND_MAX is %d\n", RAND_MAX);
//     return 0;
// }
```

3. [10] Internet IPv6 address is 128 bits, which is usually represented as eight 16-bit numbers seperated by colon (:). An example IPv6 address is 1050:0000:0000:0000:0005:0600:300c:326b, each 16-bit number is represented in the format of 4 hex digits. Please write two functions where
(a) one takes an array of 8 short int as a IPv6 address and store it in two 64-bit long int,
(b) one takes two 64-bit long int as a IPv6 address and output its colon-separated representation.
Note: The bitwise operators (such as shift, and, or) are not allowed to be used in this question.

```
//(a)   a function that takes an array of 8 short integers as
a IPv6 address and store it in two 64-bit long integers.
#include<stdio.h>

long int fucn(short int* array){
    long int temp1 = 0;
    long int temp2 = 0;
    int i ;
    for(i = 0 ; i<4 ; i++)
    {
        temp1 += array[i];
        if (i!=3){
        temp1 *= 65536;
        }
    }
    for(i=4;i<8;i++)
    {
        temp2 += array[i];
        if (i!=3){
        temp2 *= 65536;
        }
    }
    long int result[2];
    result[0] = temp1;
```

```
    result[1] = temp2;
    return result;
}
//(b)   a function that takes two 64-bit long integers as a
IPv6 address and output its colon-separated representation.
short int func2(long int* array,int len)
{
    int i;
    short int result[8];
    for(i=1;i>=0;i--)
    {
        result[4*i+3]=array[i]%65536;
        array[i]/=65536;
        result[4*i+2]=array[i]%65536;
        array[i]/=65536;
        result[4*i+1]=array[i]%65536;
        array[i]/=65536;
        result[4*i]=array[i]%65536;
        array[i]/=65536;
    }
    /*result[7]=array[i]%65536;
    array[1]/=65536;
    result[6]=array[i]%65536;
    array[1]/=65536;
    result[5]=array[i]%65536;
    array[1]/=65536;
    result[4]=array[i]%65536;
    array[1]/=65536;

    result[3]=array[i]%65536;
    array[0]/=65536;
    result[2]=array[i]%65536;
    array[0]/=65536;
    result[1]=array[i]%65536;
    array[0]/=65536;
    result[0]=array[i]%65536;
    array[0]/=65536;*/
}
```

4. [10] When we use printf(…) function, why the "#include <stdio.h>" is needed and where, in the LINUX file system, you can locate file stdio.h? And Explain the differences between #include <my_file.h> and #include "my_file.h"

   printf 相關的定義與宣告都在stdio.h內，在Unix/Linux系統中，通常位於 /usr/include，<my_file.h> 是在系統目錄中尋找並Include my_file.h檔案，"my_file.h"則是在當前目錄下尋找該檔案

5. [5] What is the outputs of the following statements:
   (a)for (int i = 0; i < 10, i != 3, i != 5; i++) printf("%d\n", i);
   (b)for (int i = 0; i < 10, i = 3, i != 5; i++) printf("%d\n", i);

   (a)
   0
   1
   2
   3

4
(b)
3 (Infinite loop)

6. [5] Write iterative and recursive versions of the sequence numbers defined by the following:
   (1) A(n) = output (n) + B(n – 1) and A(0) = output (0);
   (2) B(n) = output (n) + A(n/2) and B(0) = output (0);
   (3) What is the outputs of B(20).

(a)

recursive:

```
int A(int n){
   if(n == 0){
      return 0;
   }
   return ((n + n - 1) + A((n - 1) / 2));
}
```

```
int B(int n){
   if(n == 0){
      return 0;
   }
   return (n + (n/2) + B((n/2)-1));
}
```

Iterative:

```
int A(int n){
   int sum = 0;
   while(n != 0){
      sum = sum + 2 * n - 1;
      n = n / 2;
   }
   return sum;
}
```

```
int B(int n){
   int sum = 0;
   while(n != 0){
      sum = sum + 1.5 * n ;
      n = n / 2 - 1;
   }
   return sum;
}
```

(b)

B(20) = 47

7. [10] Write a function unsigned a_plus_b_mod_c(unsigned a, unsigned b, unsigned c) to compute (a+b)%c and a function unsigned a_times_b_mod_c(unsigned a, unsigned b, unsigned c) to compute (a*b)%c, where a, b, c are numbers of type unsigned.

```
#include <stdint.h>
#include <stdio.h>
uint8_t apbmc(uint8_t a, uint8_t b, uint8_t c) {
  a = a%c;
  b = b%c;
  // c > a > b; c > b > a; c > a = b;
  if (b > a) {
    printf("b > a\n");
    if (a < (c - b)) {
      // a < (c - b), (a + b) < c
      return (a + b) % c;
    }
    return (a - (c - b)) % c;
  } else {
    printf("a >= b\n");
    if (b < (c - a)) {
      // a < (c - b), (a + b) < c
      return (a + b) % c;
    }
    return (b - (c - a)) % c;
```

```
    }
}
```

8. [10] #define N 10
int A[N][N], B[N][N];
Write a function that can compute the matrix multiplication of two 2-D N×N integer matrices in which it must call another function to compute inner product of a row in A and a column in B that uses the declaration of int (*p)[N].

```
int innerProduct(int rowA[N], int (*p)[N]) {
    int result = 0;
    for (int i = 0; i < N; ++i) {
        result += rowA[i] * **(p + i);
    }
    return result;
}

void matrixMultiply(int A[N][N], int B[N][N], int
result[N][N]) {
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            result[i][j] = innerProduct(A[i], (int(*)[N])
&(B[0][j]));
        }
    }
}
```

9. [10] Write a function that uses a 1-D array to store the Pascal's triangle.

Example:
```
          1.....(First line)
        1   1.....(Second line)
      1   2   1.....(Third line)
    1   3   3   1          .
  1   4   6   4   1        .
1   5  10  10   5   1      .
  1   6  15  20  15   6   1    .
1   7 21  35  35  21  7   1   .
```

1-D array should store the content: 1 1 1 1 2 1 1 3 3 1 1 4 6 4 1 1 5 10 10 5 1 .....
Write another function as follows: The function returns the value based on the row and col.
int Pascal(int row, int col)
{ ...}
Ex: Pascal(5,3), it returns 6.

```
#include <stdio.h>

int pascal (int row, int col){
    if((col - 1) == 0 || col == row){
        return 1;
    }
    else{
        return pascal(row - 1, col - 1) + pascal(row - 1,
col);
    }
}
```

```c
void triangle(int deep){
    int size = (1 + deep)* deep / 2 ;
    int arr[size];
    for(int i = 0; i < deep; i++){
        for(int j = 0; j <= i ; j++){
            arr[(1 + i) * i / 2 + j] = pascal(i + 1, j + 1);
        }
    }
    for(int i = 0; i < size; i++){
        printf(" %d", arr[i]);
    }
    printf("\n");
}


int main(){
    int x , y, res, d;
    scanf("%d", &d);
    triangle(d);
    scanf("%d %d", &x, &y);
    res = pascal(x,y);
    printf("%d\n", res);
    return 0;
}
```

10. [10] (a) Write a recursive function *f(int n, int m)* that can print m^n combinations of v[1] v[2] … v[n], where v[i] ∈ {1, 2, …, *m*}. For example, if *n* = 2 and *m* = 3, the following number list is printed: ***1***
1, 1 2, 1 3, 2 1, 2 2, 2 3, 3 1, 3 2, 3 3
(b) Modify *f(int n, int m)* to print the combinations of v[1] v[2] … v[n], where v[i] ∈ {1, 2, …, m} and v[1]≠v[2] ≠…≠v[n]. So, if n = 2 and m = 3, the following number list is printed:
1 2, 1 3, 2 1, 2 3, 3 1, 3 2

```c
#include <stdio.h>
#include <stdlib.h>

int *v;
int index = 0;
int firstSet = 1;

void f(int n, int m) {
    if (index == n) {
        if (!firstSet) {
            printf(",");
        }
        for (int i = 0; i < n; i++) {
            printf(" %d", v[i]);
        }
        firstSet = 0;
        return;
    }

    for (int i = 1; i <= m; i++) {
        int x = 1;
        for(int j = index; j > 0 ; j--){
```

```
                if (v[j - 1] == i){
                    x = -1;
                }
            }
            if(x < 0) continue;
            v[index] = i;
            index++;
            f(n, m);
            index--;
        }
    }

    int main() {
        int n, m;
        scanf("%d %d", &n, &m);
        v = (int*)malloc(n* sizeof(int));
        f(n, m);
        free(v);
        return 0;
    }
```

11. [10] The syntax of *switch* statement is defined formally in BNF form as follows:

switch_statement :: = switch (*integral_expression*)
{case_statement | default_statement | switch_block}$_{1}$

case_statement :: = *{* case *constant_integral_expression* : *}*$_{1+}$ *statement*

default_statement :: = default : *statement*

switch_block ::= *{{declaration_list}*$_{opt}$ *{case_group}*$_{0+}$ *{default_group}*$_{opt}$ *}*

case_group ::= *{* case *constant_integral_expression :}*$_{1+}$ *{ statement }*$_{1+}$

default_group ::= default : *{statement}*$_{1+}$

Note: *{A}*$_{1}$ means A must appear only once, *{A}*$_{+0}$ means A must appear 0 or more times, *{A}*$_{+1}$ means A must appear at least once, and *{A}*$_{opt}$ means A is optional. Based on the above BNF definition of switch, are the following statements legal? Explain.

int i; float f=2.0;
(1) default: printf("Ex1: default\n");
(2) case : printf("Ex1: default\n");
(3) switch (i) case i: printf("case 1\n");
(4) switch (i) default: printf("Ex1: default\n");
(5) switch (i) {}
(6) switch (f) {}
(7) switch (i) if(1) printf("switch includes if statement\n");
(8) switch (i) printf("switch includes if statement\n");
(9) switch (i) {
int x,y;
case 1: x=10; y=20; printf("x=%d, y=%d\n", x, y); break;
default: x=2; printf("default, x=%d\n", x);
}
(10) switch (i) {
i = printf("default\n");
default: printf("i=%d\n", i);
}

(1) Illegal, no switch statement
(2) Illegal, no switch statement
(3) Illegal, the “i” after the “case” is not a constant_integral_expression
(4) Legal
(5) Legal
(6) Illegal, "f" is a float, not a constant_integral_expression
(7) Illegal, switch statement should be followed by a switch block
(8) Illegal, switch statement should be followed by a switch block
(9) Legal
(10) Legal

12. [10] (A)
int a[] = {5, 15, 20, 25, 30, 35, 40, 45};
int *p = &a[1], *q = &a[5]; write your result and explain.
(a) What is the value of q-p?
(b) Is the condition p < q true or false?
(c) Is the condition *p < *q true or false?
(d) int (*p)[10], *q[10], **r; Are p, q, and r equivalent?
(e) int i, *p = &i, **q = &p, ***r = &q; (note: you need to draw a figure to answer this one)

(B)
What are the outputs of the following program? Please find out if there is any errors.

```
int main(int argc, char *argv[])
{
 int a[10]={0, 1,2,3,4,5,6,7,8,9};
 int *p=a, i=0, j=10;
 int c;

   printf("p=%x, *p=%d\n", p,*p);
   printf("*p++=%d\n", *p++);
   printf("p=%x, *p=%d\n\n", p,*p);
 p=a;
   printf("p=%x, *p=%d\n", p,*p);
   printf("*(p++)=%d\n", *(p++));
   printf("p=%x, *p=%d\n\n", p,*p);
c = *p++;
   printf("c=%d\n", c);
   printf("++i-- = %d\n", ++i--);
   ++i = 2;

 i=0; j = 10;
   printf("j+++i = %d\n", j+++i);
   printf("i = %d, j = %d\n", i, j);
 i=0; j = 10;
   printf("j+++i-- = %d\n", j+++i--);
   printf("i = %d, j = %d\n", i, j);
}
```

(A)
int a[] = {5, 15, 20, 25, 30, 35, 40, 45};
int *p = &a[1], *q = &a[5]; write your result and explain.
(a) What is the value of q-p? Ans:4

(b) Is the condition p < q true or false? Ans:True
(c) Is the condition *p < *q true or false? Ans:True
(d) int (*p)[10], *q[10], **r; Are p, q, and r equivalent? Ans:False
(e) int i, *p = &i, **q = &p, ***r = &q; (note: you need to draw a figure to answer this one)
Ans:[i] <-p <- q <- r

(B)

```
int main(int argc, char *argv[])
{
 int a[10]={0, 1,2,3,4,5,6,7,8,9};
 int *p=a, i=0, j=10;
 int c;

printf("p=%x, *p=%d\n", p,*p);      //p=(address of a[0]), *p=0
 printf("*p++=%d\n", *p++);        //*p++=0
 printf("p=%x, *p=%d\n\n", p,*p);   //p=(address of a[1]), *p=1

 p=a;
printf("p=%x, *p=%d\n", p,*p);      //p=(address of a[0]), *p=0
 printf("*(p++)=%d\n", *(p++));     //*(p++)=0
 printf("p=%x, *p=%d\n\n", p,*p);   //p=(address of a[1]), *p=1

c = *p++;
 printf("c=%d\n", c);              //c=1
 printf("\n");

//printf("++i-- = %d\n", ++i--);     //  This part wrong
//++i = 2;

 i=0; j = 10;
 printf("j+++i = %d\n", j+++i);     //j+++i = 10
 printf("i = %d, j = %d\n", i, j);      //i = 0, j = 1
 i=0; j = 10;
 printf("j+++i-- = %d\n", j+++i--); //j+++i-- = 10
 printf("i = %d, j = %d\n", i, j);      //i = -1, j = 11
}
```

13. [10] Write a function *poly* using va_list (similar to printf(…)) to compute a polynomial of $A_n x^n + A_{n-1}x^{n-1} + A_{n-2}x^{n-2} + \ldots + A_1 x^1 + A_0$. So, you can call poly(5, 3.2, 3.0, 2.0, –5.0, 0.0, 7.0, –6.0) as an example to compute $3.0\times(3.2)^5 + 2.0\times(3.2)^4 - 5.0\times(3.2)^3 + 7.0\times(3.2) - 6.0$.

```
double poly(int num, ...){
    va_list args;
    double sum = 0;
    int t = num;
    num += 1;
    va_start(args, num + 1);
    double X = va_arg(args, double);

    for(int i = 0 ; i < num ; i++)
        sum += va_arg(args, double) * pow(X, t--);
```

```
        return sum;
    }
```

14. [10] Please trace the following code and get the results in *n*, *L*[], and *U*[] by using [*Rmin*, *Rmax*] = [333, 333] and [3, 128]. Assume *powerof2*[*i*] stores 2^i for i = 0 to 31, and W = 32.

```
void Direct_convert(unsigned Rmin, unsigned Rmax, unsigned L[], unsigned U[], unsigned *n)
{
   int d = 0, i;
   while (Rmin <= Rmax) {
       for (i = 0; i < W; i++)
           if ((Rmin%powerof2[i+1] != 0) || (Rmin + powerof2[i+1] – 1) > Rmax) break;
      L[d] = Rmin;      U[d] = Rmin + powerof2[i] – 1; d = d + 1;
      Rmin = Rmin + powerof2[i];
   }
   *n = d;
}
```

(1)
n = 1
L[] = {333}
U[] = {333}

(2)
n = 7
L[] = { 3, 4, 8, 16, 32, 64, 128}
U[] = { 3, 7, 15, 31, 63, 127, 128}