1141 NTOUCSE 程式設計 1C Midterm

Name: ______________ Major:______________ Student Id:______________ **114/10/28** (二)

Exam Time: **13:20 – 16:00**

Exam rules： 1. **Close book**, **close** everything including quizzes, homeworks, assignments, reference materials, etc.
2. You can answer the questions in **English** or in **Chinese**, in this **problem sheet**, the **answer sheet**, or **both**.
3. You can use language features not taught in class if you feel necessary, but strictly limited in **C**.
4. **No** mobile phone, pad, computer or calculator is allowed. (Electronic) English dictionary is OK
4. No peeping around! No discussion! No exchange of any material; **raise your hand if you have any question about the exam problems**
5. If you turn in the paper earlier than the specified time, **leave the classroom immediately and quietly**
6. Against any of the above rules will be treated as cheating in the exam and handled by school regulations.
7. **Turn in BOTH the problem sheet with your name and id and answer sheet with your name and id**.

1. (a) [10] Please complete the program (shown on the right) to read multiple lines of data until the end of the data stream, as shown in the figure below. The first character of each line is 'O', 'H', or 'D', representing an integer that is:Octal with 13 to 18 digits, Hexadecimal with 10 to 14 digits, or Decimal with 13 to 17 digits, respectively. The number inside the parentheses is also of the same radix (base) and within the same range as the preceding number, and there are no spaces inside the parentheses. Please print the difference between the two numbers in that line in decimal, left-aligned in a 20-character space. In the figure on the bottom right, □ represents a space. Please define two variables, x and y, using appropriate data types, and use scanf to read the data and printf to output the difference between the two numbers. (If your answer contains necessary spaces, please clearly indicate them with □.)

```
01 #include <__________>
02
03 int main()
04 {
05         ________ x, y;
06         char c;
07         while (1==scanf("______", ______))
08         {
09                 if (__________)
10                         scanf("__________", &x, &y);
11                 else if (__________)
12                         scanf("__________", &x, &y);
13                 else
14                         scanf("__________", &x, &y);
15                 printf("_____________\n", x-y);
16         }
17         return 0;
18 }
```

```
O12345676544012□□□(725571123)↵
X4AF31024FC□(2451)↵
D44890231002(9201235)↵
D12139876□□(22456678)↵
```

(b) [6] If the above program uses a character array to store the format string, lines 05-14 can be modified as follows:

```
06        char c, fmt[]=-------------; // the same as the format string in line 10, 12, or 14 of part(a)
07        while (1==scanf(---------------------)) // the same as line 07 of part (a)
08        {
09                fmt[________] = fmt[________] = _______;
10                scanf(________, &x, &y);
```

(c) [4] Please complete the program on the right: using printf() to

```
000077,□□□□□5.68
```

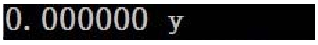
obtain the output shown on the left. The output should consist of:a 6-digit octal integer, padded with leading zeros if it has fewer than 6 digits (no spaces), followed by a comma, followed by a decimal floating-point number (6 digits for the integer part and 2 digits for the fractional part, with the

```
01 int x=63;
02 double y=5.678;
03 printf("______________\n", x, y);
```

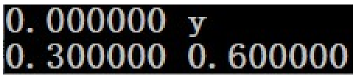
third decimal place and beyond rounded).In the above figure, □ represents a space.

```
double x;
for (x=0; x!=3000; x+=0.3)
{
        printf("%f □", x);
}
```

(d) [10] Sometimes, to find logical errors in your program, you print the values of some variables inside a loop. However, if the loop repeats many times, the screen scrolls too fast to analyze the printed data. If you want the user to control whether to continue printing after each output by pressing a key, for example, the program on the right is found to be running endlessly, so you add a getchar() statement right after printf() statement. Now the program pauses after the first data 0.000000 is printed and you press a ‘y’ as shown in the figure on the right. But the loop does not continue at this time unless you press a <enter>. Now the program prints two numbers at a time, as shown in the figure on the right. How does this happen? How can you press just one key to display one data without modifying the program at this time? How should the program be modified if you want the program to terminate when the 'a' key is pressed?



```
0.000000 y
```



```
0.000000 y
0.300000 0.600000
```

2. The objective of the following several versions of program is exactly the same: to read an alphanumeric string without spaces from the keyboard, for example, **xndrf**, and then print the string rotated starting from a different character, as shown in the figure on the left. Please complete the program for each of the following sub-questions according to the individual requirements.

```
xndrf↵
ndrfx↵
drfxn↵
rfxnd↵
fxndr↵
```

```
01 #include <stdio.h>
02 int main()
03 {
04        char_________;
05        int i, j, len;
06        while (___==scanf("%___%n",  ___, &len))
07              for (i=0; i<len; i++)
08              {
09                    for (j=0; j<len; j++)
10                          printf("%c", buf[____%___]);
11                    printf("\n");
12              }
13        return 0;
14 }
```

(a) [10] Complete the program shown in the upper right figure, line 04 defines a **char** array **buf** capable of holding up to **1000** characters. Line 06 uses the scanf utility to read the input string into the array **buf** and the length of the string into the integer variable **len**. Line 10 directly calculates the position of the character in the array to be printed based on the loop control variables **i** and **j** and the string length **len**. Please note that when scanf() reads a string, it places a '\0' character after the end of the string, and the ASCII code value of this character is 0。

(b) [4] Please complete the program in the figure on the right. The main difference between this program and the program in part (a) is that lines 09-12 use two separate loops (lines 09, 10 and lines 11, 12) for each starting rotation value **i**, to print the second half and the first half of the string.

```
07          for (i=0; i<len; i++)
08          {
09                for (j=0; j<_____; j++)
10                      printf("%c", buf[_____]);
11                for (j=0; j<_____; j++)
12                      printf("%c", buf[_____]);
13                printf("\n");
14          }
```

(c) [2] In parts (a) and (b), the length of the input string, **len**, was obtained simultaneously with reading the string using scanf(). This input string length could, of course, also be calculated using an extra loop or the strlen() utility function from string.h, but it is actually not always necessary to use this length information (**len**) to control the loops in the program. Please complete the program snippet in the figure on the right to replace lines 06-08 of the program for part (a), without using the string's length to control the number of loop iterations.

```
06      while (1==scanf("%s", buf))
07          for (i=0;________; i++)
08          {
09              ...
```

(d) [6] In parts (a) and (b), the output was done character by character using printf("%c", ...), however, printf() can output an entire string at once. Please complete the program snippet in the figure on the right, using two printf() statements to replace the output loop on lines 09 and 10 in part (a), or the two output loops on lines 09-12 in part (b). Note that line 11 in the figure requires modifying the **i**-th element of the **buf** character array to utilize the string printing feature of printf(). However, because lines 09-13 are inside the loop starting at line 07, if the buf character array is corrupted, the subsequent loop iterations after **i**++ will not work correctly. Therefore, line 10 is required to first back up the content of **buf[i]** into the character variable **t**, and restores it in line 13.

```
07      for (i=0; i<len; i++)
08      {
09          printf("________",________);
10          t = buf[i];
11          buf[i] = ________;
12          printf("________\n",________);
13          ________;
14      }
```

(e) [4] In the previous sub-questions (a) through (d), loops were used to fulfill the problem's requirements. We know that recursive functions are another programming construct for expressing repetitive actions, and recursive designs can be quite varied. The program snippet on the right is a recursive design. Lines 16-20 contain a loop within the main() function that calls the recursive function, replacing the functionality of lines 07-12 in question (a). The parameter **i** of the recursive function on line 02 specifies that the printing for this call should start from the **i**-th character of the **buf** array, and the parameter **c** denotes the number of characters to be printed. Please complete the program snippet. (Hint: The concept of recursion is that after printing the **i**-th character, the task is completed by simply printing the remaining part of the string starting from the next character.)

```
16          for (i=0; i<len; i++)
17          {
18              rotate_print1(buf, i, len);
19              printf("\n");
20          }
```

```
02 void rotate_print1(char buf[], int i, int c)
03 {
04      if (c>0)
05      {
06          printf("%c", buf[i]?buf[i]:buf[i=0]);
07          rotate_print1(buf, ______, ______);
08      }
09 }
```

(f) [4] The program snippet on the right, lines 21-25, contains a loop within the main() function that calls the recursive function, replacing the functionality of lines 07-12 in question (a). The parameter **i** of the recursive

```
21          for (i=0; buf[i]!=0; i++)
22          {
23              rotate_print2(buf, i, 0);
24              printf("\n");
25          }
```

```
02 void rotate_print2(char buf[], int i, int j)
03 {
04      if (j<i)
05      {
06          rotate_print2(buf, i, j+1);
07          printf("%c", buf[______]);
08      }
09      else if (buf[j]!=0)
10      {
11          printf("%c", buf[______]);
12          rotate_print2(buf, i, j+1);
13      }
14 }
```

function on line 02 specifies which character the current print operation should start from. Each recursive call only prints one character. The parameter **j** represents the index of the recursive call. Please complete the program snippet. (Hint: When **j** < **i**, you need to find out the index of the **buf** array for the last **j**-th character output.)

3. If **x** is a **double** variable and **n** is a non-negative integer, it is extremely inefficient to calculate x^n with **n-1** times of multiplication as x * x * … * x. You need to use "square and multiply" strategy to implement an efficient program. There are two ways to implement the "square and multiply" strategy. For example, $x^{13}=x^{2^3+2^2+0\cdot 2^1+1} = x^{\underline{\underline{\underline{((\underline{1})\cdot 2+1)}\cdot 2+0)}\cdot 2+1}} = \underline{\left(\underline{\left(\underline{(\underline{x})^2\cdot x}\right)^2\cdot x^0}\right)^2\cdot x}$ or

$$x^{13}=x^{1+0\cdot 2^1+2^2+2^3}=(x)^1\cdot(x^2)^0\cdot(x^4)^1\cdot(x^8)^1$$

(a) [10] Please complete the function in the figure on the bottom right, which implements the first calculation method. Since this method starts calculating from the most significant bit of **n**, the two while loops on lines 04 and 05 first calculate **w** = $\mathbf{2^{m-1}}$, where **m** is the number of bits in **n**. Lines 06-10 use **w** to control the number of loop iterations. Line 8 first squares the current product. Line 9 then decides, based on whether the corresponding bit of **n** is 0 or 1, whether to subtract **w** from **n** and multiply the current product by **x**

```
01 double power1(double x, int n)
02 {
03        double prod;
04        int n1=n, w=________;
05        while ((n1/=2)>0) ________;
06        for (prod=1; w>0; w/=2)
07        {
08               _______________;
09               if (________)
10               {
11                      n-=w;
12                      _______;
13               }
14        }
15        return prod;
16 }
```

(b) [6] Please complete the recursive function in the figure below to implement the algorithm in part (a)

```
01 double power1(double x, int n)
02 {
03        if (n==0)
04               return 1.;
05        else
06        {
07               double x2 = power1(_____, _____);
08               return (_____?_____:_____) * _____;
09        }
10 }
```

(c) [8] Please complete the function below to implement the second calculation method, where the loop on lines 04 and 05 repeats for a number of times equal to the number of binary bits in **n** controlled by the value of **n** itself, dividing by **2** each time until it reaches **0**. The value of **x** in each loop iteration is the square of its value from the previous iteration.

```
01 double power2(double x, int n)
02 {
03        double prod;
04        for (prod=1; n_____; n_____)
05        {
06               prod *= ________;
07               x _______;
08        }
09        return prod;
}
```

(d) [6] Please complete the recursive function below to implement the algorithm in part (c)

```
double power2(double x, int n)
{
    if (n==0)
        return 1.;
    else
        return _______* power2(_______, _______);
}
```

4. (a) [4] The **double** array **values** in the program below holds **n** floating numbers. In line 08, we plan to utilize the qsort() utility in the stdlib library to put these data in ascending order. Please complete this program:

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 int compare(const void *pa, const void *pb) { return *(double *)pa-*(double *)pb; }
04 int main()
05 {
06      int i, n=10;
07      double values[] = {40.3, 10.2, 10.8, 10.4, 11.1, 10.2, 100.9, 90.1, 20.2, 25.4};
08      qsort(________,________,________,________);
09      for (i=0; i<n; i++)
10          printf("%6.1f ", values[i]);
11      printf("\n");
12      return 0;
13 }
```

(b) [2] The prototype of the utility function qsort() is shown as follows. What do we usually call the type of the fourth parameter?

void qsort(void* base, size_t num, size_t size, int (***compare**)(const void*,const void*));

(c) [2] The output of the above program is as follows:

```
10.2   10.8   10.4   11.1   10.2   20.2   25.4   40.3   90.1  100.9
```

The order is clearly incorrect, not ascending. What is the problem with the above program?

(d) [2] Please finish the following correction to the above program.

```
int compare(const void *pa, const void *pb)
{
    double *pa1 = ________pa, *pb1 = ________pb;
    if (*pa1 < *pb1)
        return ________;
    else if (*pa1 == *pb1)
        return ________;
    else
        return ________;
}
```