The Programmer's Way to C

http://cafe.daum.net/pway

Chapter 1. Choosing A Language

The language we use most at Programmer's Way is C. C has both a prominent place in programming history and a notorious reputation among beginning programmers.

Although C is not the easiest language to learn, one cannot study programming without it. Languages such as C++, Java, C# and many more share a structure similar to C.

The difficulty encountered learning C is usually because teachers push beginning students beyond their capability.

Actually, only a few points are comprehendable to novices in C.

1. Printf functions
2. Simple variables
3. The for statement

One can create complex programs using only these.

Chapter 2. Before starting the First Problems

2.1 The First Program

1. Execute the following program.

```
#include<stdio.h>
int main()
{
    printf("This is my first program\n");
    return 0;
}
```

2. Let's make it a little more complex.

```
#include<stdio.h>
int main()
{
    printf("\n");
    printf("-------------------------\n");
    printf("This is my second program\n");
    printf("-------------------------\n");
    return 0;
}
```

It is imperative to get the hang of printf functions. With printf functions you can finally *show* something.

2.2 The Second Program

1. Execute the following program.

```
#include<stdio.h>
int main()
{
    int a, b, c;

    a = 1;
    b = 2;
    c = a + b;

    printf("a=%d b=%d c=%d\n", a, b, c);
    return 0;
}
```

The concept of variables is essential in writing programs. In the program above a, b and c are variables. Variables

1. Have names.

2. Have definite forms to their data.

3. Exist in the computer’s memory.

(Have a place and size.)

4. Can be used to store data.

5. One can use data that has been kept in variables.

Such are the features of variables.

2.3 The Third Program

1. Execute the following program.

```
#include<stdio.h>
int main()
{
    int a, b, c;

    printf("please enter two numbers:");
    scanf("%d %d", &a, &b);

    c = a + b;
    printf("\na=%d b=%d c=%d", a, b, c);
    return 0;
}
```

Printf functions are used for output. The first input function to learn is the scanf function. In scanf functions, one must insert the character & in front of the receiptant variable, in the above case a, b and c. This is a point to ponder later on; let it drop at the moment.

2.4 The Fourth Program

1. Execute the following program.

```
#include<stdio.h>
int main()
{
    int i, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);
    printf("\n");
    for(i=0;i<n;i++)
        printf("yes ");
    printf("\n print %d many yes",n);
    return 0;
}
```

A slightly complicated program, basic for understanding the for statement.

2. Execute the following program.

```
#include<stdio.h>
int main()
{
    int i, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

    printf("\n");
    for(i=0;i<n;i++)
        printf("%d", i);
    printf("\n print the numbers from 0 to %d", n-1);
    return 0;
}
```

2.4.1 Understanding the For Statement Pt.1

Besides printf and scanf functions, the first friend we meet in the world of programming is the for statement.

The for statement is a program control structure used in finite repetitive cycles.

Output "yes " ten times.

Output "happy " one hundred times.

Output "laugh " one thousand times.

The program control structure used in such cases is the for statement.

```
    for(i=0;i<10;i++)
        printf("yes ");

    for(i=0;i<100;i++)
        printf("happy ");

    for(i=0;i<1000;i++)
        printf("laugh ");
```

2.4.2 Understanding the For Statement Pt.2

The for statement is used in finite repetitive cycles; sometimes a variable may be given instead of a number.

Consider the following case.

```
    for(i=0;i<n;i++)
        printf("yes ");
```

The number of "yes " outputted will vary according to the value of n. To better understand how the for statement works, let us take a simple example. Suppose the value of n is 3.

The for statement given above will execute in the following order.

```
 0. i=0;
 1. Test i<n; 0<3 , True.
 2. printf("yes ");
 3. i++   i increase by 1, i=1
 4  Test i<n;  1<3 , True.
 5. printf("yes ");
 6. i++   i increase by 1, i=2
 7. Test i<n;  2<3 , True.
 8. printf("yes ");
 9. i++   i increase by 1, i=3
10. Test i<n;  3<3 , False.
11. Close for statement.
```

2.4.3 Understanding the For Statement Pt.3

Shall we go a little deeper into the for statement?

```
    for(i=0;i<n;i++)
        printf("yes ");
```

The for statement’s structure is something like this.

```
    for(initialize; execute condition; increase variable)
        execution statement
```

In the above, i=0 is the initialization. This initialization is used only once when executing a for statement, at the start.

i<n is the execute condition. If the execute condition is correct, the execution statement is exectuted and the increase variable is executed. After that (execute condition-> execution statement-> increase variable->)is repeated. If the exectution condition is incorrect the for statement closes.

The execution statement above is printf("yes "). i++ is the increase variable.

Remember that there is only one exectution statement in the for statement.

2.4.4 Understanding the For Statement Pt.4

One can use variables to control for statements inside of for statements.

```
    for(i=0;i<n;i++)
        printf("%d", i);
```

Suppose the value of n is 3.

The for statement given above will execute in the following order.

```
 0. i=0;
 1. Test i<n;  0<3 , True.
 2. Output printf("%d", i);
         0
 3. i++   i increase by 1, i=1
 4  Test i<n;  1<3 , True.
 5. Output printf("%d", i);
         1
 6. i++   i increase by 1, i=2
 7. Test i<n;  2<3 , True.
 8. Output printf("%d", i);
         2
 9. i++   i increase by 1, i=3
10. Test i<n;  3<3 , False.
11. Close for statement.
```

Remember that the value of i is 3 at the end of the for statement.

2.5 The Fifth Program

1. Execute the following program.

```
#include<stdio.h>
int main()
{
    int i, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

    for(i=0;i<n;i++)
        printf("%d", i+1);
    return 0;
}
```

2. In the above program, substitute

`printf("%d\n", i+1);  for`

`printf("%d", i+1);`

3. Execute the following program.

```
#include<stdio.h>
int main()
{
    int i, j, n;
    printf("\nPlease enter number:");
    scanf("%d", &n);

    for(i=0;i<n;i++)
    {
         for(j=0;j<n;j++)
             printf("%4d", j+1);
         printf("\n");
    }
    return 0;
}
```

What do you mean, shoving a program this complicated in my face! Sadly, this is the path we are destined to tread.

2.5.1 Understanding the For Statement Pt.5

Let us review the program given in the Fifth Program.

```
 1.  #include<stdio.h>
 2.  int main()
 3.  {
 4.      int i, j, n;
 5.
 6.      printf("\nPlease enter number:");
 7.      scanf("%d", &n);
 9.
10.      for(i=0;i<n;i++)
11.      {
12.           for(j=0;j<n;j++)
13.               printf("%4d", j+1);
14.           printf("\n");
15.      }
16.      return 0;
17.  }
```

I have numbered the lines for explanation.

Question: You said there was only one execution statement in the for statement. Why do I see lots of them?

Answer: The execution statement for line 10 `for(i=0;i<n;i++)` is from line 11 to line 15.

Question: You're saying that the complicated statements on line 5 are really one statement?

Answer: Yes. C treats compound statements (or blocks) as a single statement. Compound statements let you bundle lots of statements into one.

Question: Can I use a for statement inside a compound statement?

Answer: Yes. You can put anything, no matter how complicated, in compound statements. The most complicated statement we have encountered so far is the for statement. Of course it can go inside a compound statement.

Question: When do you think I'll actually understand this stuff?

Answer: Anybody can. You just need a little imagination.

2.5.2 Understanding the for statement Pt.6

Review the program we saw in the Fifth Program.

```
 1.  #include<stdio.h>
 2.  int main()
 3.  {
 4.      int i, j, n;
 5.
 6.      printf("\nPlease enter number:");
 7.      scanf("%d", &n);
 9.
10.      for(i=0;i<n;i++)
11.      {
12.          for(j=0;j<n;j++)
13.              printf("%4d", j+1);
14.          printf("\n");
15.      }
16.      return 0;
17.  }
```

Let's see how this program works. Suppose the value of n is 3.

In the for statement of line 10

```
initialize:    i=0              i becomes 0
execute condition:  i<3
execution statement:    execute lines 11 to 15
      In the for statement of line 12
          initialize:   j=0                   j becomes 0
          execute condition: j<3
          execution statement:   printf("%4d", j+1);   output 1
          increase variable: j++                    j becomes 1
          execute condition: j<3
          execution statement:   printf("%4d", j+1);   output 2
          increase variable: j++                   j becomes 2
          execute condition: j<3
          execution statement:   printf("%4d", j+1);   output 3
          increase variable: j++                   j becomes 3
          execute condition  j<3
          close for statement(line 12).
      In line 14
          printf("\n");            output line switch
      Close statement(lines 11 to 15)
increase variable:  i++              i  becomes 1
execute condition:  i<3
execution statement:    executes line 11 to 15
      In the for statement of line 12
          initialize:   j=0                   j becomes 0
          execute condition: j<3
          execution statement:   printf("%4d", j+1);   output 1
          increase variable: j++                   j becomes 1
          execute condition: j<3
```

```
            execution statement:   printf("%4d", j+1);   output 2
            increase variable: j++                     j becomes 2
            execute condition: j<3
            execution statement:   printf("%4d", j+1);   output 3
            increase variable: j++                     j becomes 3
            execute condition  j<3
            Close for statement(line 12)
        In line 14
            printf("\n");                    Output line switch
        Close statement(lines 11 to 15)
    increase variable:  i++              I becomes 2
    execute condition:  i<3
    execution statement:    execute lines 11 to 15
        In the for statement of line 12
            initialize:   j=0                   j becomes 0
            execute condition: j<3
            execution statement:   printf("%4d", j+1);   output 1
            increase variable: j++                     j becomes 1
            execute condition: j<3
            execution statement:   printf("%4d", j+1);   output 2
            increase variable: j++                     j becomes 2
            execute condition: j<3
            execution statement:   printf("%4d", j+1);   output 3
            increase variable: j++                     j becomes 3
            execute conditon  j<3
            Close for statement (line 12).
        In line 14
            printf("\n");                    Output line switch
        Close statement(lines 11 to 15)
    increase variable:  i++              i  becomes 3
    execute condition:  i<3
    Close for statement (line 10).
```

Therefore

```
   1   2   3
   1   2   3
   1   2   3
```

This output is shown onscreen.

2.6 The Sixth Program

1. Execute the following program.

```
#include<stdio.h>
int main()
{
    int i, j, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

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

```
             printf("%4d", j+1);
         printf("\n");
    }
    return 0;
}
```

2. Instead of line 12 printf("%4d", j+1); insert printf("%4d", i+1); and execute the program.

3. Execute the program inserting printf("%4d", i+j+1); in line 12.

4. Execute the program inserting printf("%4d", (i+1)*(j+1)); in line 12.

If you can guess the outcome before executing the program, your grasp of the for statement has improved.

2.7 The Seventh Program

1. Execute the following program.

```
#include<stdio.h>
int main()
{
    int i, j, k, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

    k = 1;
    for(i=0;i<n;i++)
    {
         for(j=0;j<n;j++)
             printf("%4d", k++);
         printf("\n");
    }
    return 0;
}
```

Can you guess the outcome of this program? If not, think again. Go back over how the for statement works.

2.8 The Eighth Program

1. Execute the follwing program.

```
#include<stdio.h>
int main()
{
    int i, j, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

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

```
            printf("%4d", j+1);
        printf("\n");
    }
    return 0;
}
```

2. Execute the following program.

```
#include<stdio.h>
int main()
{
    int i, j, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

    for(i=0;i<n;i++)
    {
         for(j=0;j<n-i;j++)
             printf("%4d", j+1);
         printf("\n");
    }
    return 0;
}
```

Can you guess the outcome of this program?

2.8.1 Understanding the For Statement Pt.7

Let's look at this program that appeared in the Seventh Program.

```
 1.  #include<stdio.h>
 2.  int main()
 3.  {
 4.      int i, j, k, n;
 5.
 6.      printf("\nPlease enter number:");
 7.      scanf("%d", &n);
 9.
10.      k=1;
11.      for(i=0;i<n;i++)
12.      {
13.           for(j=0;j<n;j++)
14.               printf("%4d", k++);
15.           printf("\n");
16.      }
17.      return 0;
18.  }
```

How does this program work? Suppose the value of n is 3.

```
In line 10    k=1;            k becomes 1
In the for statement of line 11
    Initialize:    i=0            i becomes 0
    execute condition:  i<3
    execution statement:    Execute 12 to 16
```

```
          In the for statement of line 13
              initialize:   j=0                   j becomes 0
              execute condition: j<3
              execution statement:   printf("%4d", k++);   output 1  k
becomes 2
              increase variable: j++                   j becomes 1
              exeucute conditon: j<3
              execution statement:   printf("%4d", k++);   output 2 k
becomes 3
              increase variable: j++                   j becomes 2
              execute condition: j<3
              execution statement:   printf("%4d", k++);   output 3  k
becomes 4
              increase variable: j++                   j becomes 3
              execution statement: j<3
              Close for statement(line 13).
          In line 15
              printf("\n");                  Output line switch
          Close statement(lines 12 to 16)
    increase variable:  i++              i becomes 1
    execute condition:  i<3
    execution statement:    Execute lines 12 to 16
          In the for statement of line 13
              initialize:   j=0                   j becomes 0
              execute condition: j<3
              execution statement:   printf("%4d", k++);   output 4  k
becomes 5
              increase variable: j++                   j becomes 1
              execute condition: j<3
              execution statement:   printf("%4d", k++);   output 5  k
becomes 6
              increase variable: j++                   j becomes 2
              execute condition: j<3
              execution statement:   printf("%4d", k++);   output 6  k
becomes 7
              increase variable: j++                   j becomes 3
              execute condition:  j<3
              Close for statement(line 13).
          In line 15
              printf("\n");                  Output line switch
          Close statement(lines 12 to 16)
    increase variable:  i++              i becomes 2
    execute condition:  i<3
    execution statement:    execute lines 12 to 16
          In the for statement of line 13
              initialize:   j=0                   j becomes 0
              execute condition: j<3
              execution statement:   printf("%4d", k++);   output 7  k
becomes 8
              increase variable: j++                   j becomes 1
              execute condition: j<3
              execution statement:   printf("%4d", k++);   output 8 k
becomes 9
              increase variable: j++                   j becomes 2
              execute condition: j<3
              execution statement:   printf("%4d", k++);   output 9 k
becomes 10
              increase variable: j++                   j becomes 3
```

```
            execute condition:  j<3
            Close for statement(line 13).
        In line 15
            printf("\n");                   Output line switch
        Close statement(lines 12 to 16)
  increase variable:  i++              i becomes 3
  execute condition:  i<3
  end for statement(line 11)
```

Therefore

```
   1   2   3
   4   5   6
   7   8   9
```

This outcome is shown onscreen.

2.8.2 Understanding the For Statement Pt.8

This program appeared in the Eighth Program.

```
 1.  #include<stdio.h>
 2.  int main()
 3.  {
 4.      int i, j, n;
 5.
 6.      printf("\nPlease enter number:");
 7.      scanf("%d", &n);
 8.
 9.      for(i=0;i<n;i++)
10.      {
11.          for(j=0;j<i+1;j++)
12.              printf("%4d", j+1);
13.          printf("\n");
14.      }
15.      return 0;
16.  }
```

The execute condition of the for statement on line 11 is different from what we have seen so far. That is, it is not a fixed variable like j<n but a uses a changing variable i like j<i+1. The execute conditions we have seen in for statements so far indicate execution times. i+1 also indicates how many times the statement should be executed.

If you have followed the for statement closely you will know that the value of i changes thus: 0, 1, 2, 3, . . . , n-1 and closes the for statement when it reaches n.

Therefore the value of i+1 will be 1, 2, 3, ... , n. A useful method when varying times in the sequence of once, twice, thrice, . . . , n.

The output when n=3 is

```
   1
   1   2
   1   2   3
```

2.8.3 Understanding the For Statement Pt.9

This program is from the Eigth Program.

```
#include<stdio.h>
int main()
{
    int i, j, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

    for(i=0;i<n;i++)
    {
         for(j=0;j<n-i;j++)
             printf("%4d", j+1);
         printf("\n");
    }
    return 0;
}
```

n-i takes the value of n, n-1, n-2, ... , 3, 2, 1. A method to use when varying times in the sequence of n times, n-1 times, n-2 times, . . . , 3 times, 2 times, 1 time.

The output when n=3 is

```
   1   2   3
   1   2
   1
```

Chapter 3. Introduction to Programming Problems

The basics of the for statement down, here are a few tasks that use only the for statement.

Programming tasks mean nothing if you do not solve them on your own.

Preserverance is the programmer's way.

You progress by solving tasks on your own. Do not seek advice that would tantamount to answers.

Posting results of programming tasks at the Programmer's Way is strictly forbidden.

Chapter 4. First Collection of Problems

Problem Set 1) Write a program that will output results according to the numbers inputted.

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

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

1-3) number = 5
      1  3  5  7  9
     11 13 15 17 19
     21 23 25 27 29
     31 33 35 37 39
     41 43 45 47 49

1-4) number = 5
      1
      1  2
      1  2  3
      1  2  3  4
      1  2  3  4  5

1-5) number = 5
      1
      2  3
      4  5  6
      7  8  9 10
     11 12 13 14 15

1-6) number = 5
      1  2  3  4  5
      1  2  3  4
      1  2  3
      1  2
      1
```

```
1-7) number = 5
      1  2  3  4  5
      6  7  8  9
     10 11 12
     13 14
     15

1-8) number = 5
      1  2  3  4  5
      2  3  4  5  6
      3  4  5  6  7
      4  5  6  7  8
      5  6  7  8  9

1-9) number = 5
      1  2  3  4  5
      2  3  4  5  1
      3  4  5  1  2
      4  5  1  2  3
      5  1  2  3  4

1-10) number = 5
                  1
               2  3
            4  5  6
         7  8  9 10
     11 12 13 14 15
```

Chapter 5. Second Collection of Problems

In the second collection of tasks, write programs without using any statements or functions other than the for statement and the printf statement. Write a program without using functions to move the cursor.

Problem Set 2) Write a program that will output results according to the numbers inputted.

```
2-1) number = 5
     *****
     *****
     *****
     *****
     *****


2-2) number = 5
     *
     **
     ***
     ****
     *****


2-3) number = 5
         *
        **
       ***
      ****
     *****


2-4) number = 5
         *
        ***
       *****
      *******
     *********


2-5) number = 5
         *
        ***
       *****
      *******
     *********
      *******
       *****
        ***
         *
```

```
2-6) number = 5
         *        *
        ***      ***
       *****    *****
      *******  *******
     ******************
      *******  *******
       *****    *****
        ***      ***
         *        *
```

```
2-7) number = 5
     [ output line n, warm-up for 2-8 ]

         *        ***********        *
        ***      *************      ***
       *****    ***************    *****
      *******  *****************  *******
     *************************************
```

```
2-8) number = 5
     [ output line 2*n ]

                       *
                      ***
                     *****
                    *******
                   *********
         *        ***********        *
        ***      *************      ***
       *****    ***************    *****
      *******  *****************  *******
     *************************************
```

```
2-9) number = 5
     $$$$$$$
     $*****$
     $*****$
     $*****$
     $*****$
     $*****$
     $$$$$$$
```

```
2-10) number = 5
   [  line n+2 + line n+1 = 2*n + line 3      ]
   [  lines that have 2 *s and               ]
   [  lines that have 1 * should output differently ]

     *
     **
     *@*
     *@@*
     *@@@*
     *@@@@*
     *@@@@@*
     *@@@@*
     *@@@*
     *@@*
     *@*
     **
     *
```

Outputting columns of characters:

```
#include<stdio.h>
int main()
{
    int i, j, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            printf("*");
        printf("\n");
    }
    return 0;
}
```

Chapter 6. The Next Stage

I'm a novice starting to warm up to programming.

It's pretty fun writing programs using the for statement and the printf statement. I think I'm cut out for programming.

For likeminded persons, there is an even more interesting Programmer's Way.

The for statement's two-layered control structure is suitable for all sorts of interesting problems. However, choicer problems call for something new.

Meet the two-dimensional array, cousin to the two-layered for statement. You usually learn two-dimensional array after one-dimensional array.

But let's take both of them at once.

An array is an extended variation of a variable. An array lets you use bundles of similar variables.

The closest thing to an array in real life is an apartment. One apartment block usually has fifteen stories, with six or eight similar suites to each story.

Numbers are used to distinguish suites in an apartment. 1408 usually means number 8 on the 14th flour.

Let us put it thus:

Soojeong Apt block 101 floor 14 number 8

So Soojeong Apt block 101 number 101 would be:

Soojeong Apt block 101 floor 1 number 1

Take away the words 'floor' and 'number'.

Soogeong Apt block 101 11

Write 'data' instead of 'Soogeong Apt block 101'.

data[1][1]

Now this is two-dimensional array.

6.1 The Ninth Program

1. Execute the following program.

```
#include<stdio.h>
int main()
{
    int a[30][30];
    int i, j, k, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

    k=0;
    for(i=0;i<n;i++)
```

```
        for(j=0;j<n;j++)
            a[i][j] = k++;

    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            printf("%4d", a[i][j]);
        printf("\n");
    }
    return 0;
}
```

6.1.1 What is Two-Dimensional Array? Pt.1

Two-dimensional array has the same features as variables.

1. Two-dimensional arrays have names.
2. Their data has a definite form.
3. They exist in the computer’s memory. (They have a place and size.)
4. They can store data.
5. They can use data that has been stored.

Array also differs from variables. Different numbers are required to distinguish individual rooms inside an array. In two-dimensional array you need a floor number and a room number. In one-dimensional array only a room number is needed.

In C floor numbers and room numbers in array start from 0. If there are ten rooms, there will be ten room numbers, 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. This sequence seems familiar. It is the same sequence the variables followed in the for statement.

The for statement and array may have something in common.

Floor numbers also start from 0 and work up to 1, 2, 3, … .

This is what I said was a two-dimensional array in the Ninth Program.

int a[30][30];

The name of this array is a. Although I have named it a, name your arrays as you please. a’s form of data is integers. a has 30 floors with 30 rooms to each floor. That’s 900 rooms altogether.

I have chosen 30 floors with 30 rooms to each, as that will be enough for the programs you will write later on.

The rooms in a

```
    a[0][0], a[0][1], a[0][2], a[0][3], ,,, a[0][28], a[0][29],
    a[1][0], a[1][1], a[1][2], a[1][3], ,,, a[1][28], a[1][29],
    a[2][0], a[2][1], a[2][2], a[2][3], ,,, a[2][28], a[2][29],
   .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    a[29][0], a[29][1], a[29][2], a[29][3], ,,, a[29][28], a[29][29]
```

are in this order, on memory.

The above uses numbers to show rooms, but a program would be dull if one could only use numbers for the rooms in an array. Therefore variants such as i or j are used to show foors or rooms.

6.1.2 What is Two-Dimensional Array? Pt.2

Review the Ninth Program.

```
 1.  #include<stdio.h>
 2.  int main()
 3.  {
 4.      int a[30][30];
 5.      int i, j, k, n;
 6.
 7.      printf("\nPlease enter number:");
 8.      scanf("%d", &n);
 9.
10.      k=0;
11.      for(i=0;i<n;i++)
12.          for(j=0;j<n;j++)
13.              a[i][j] = k++;
14.
15.      for(i=0;i<n;i++)
16.      {
17.          for(j=0;j<n;j++)
18.              printf("%4d", a[i][j]);
19.          printf("\n");
20.      }
21.      return 0;
22.  }
```

From line 11 to 13, the for statement fills the array with integers. In n floors, n rooms on each floor are filled with numbers increasing by one.

The for statement from line 15 to 20 prints the numbers in the array, floor by floor.

Similarily, the two-dimensional programs you are to write are usually composed of a part to fill the array and a part to print the array.

6.2 The Tenth Program

1. Execute the following program.

```
#include<stdio.h>
int main()
{
    int a[30][30];
    int i, j, k, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            a[i][j]=j+1;    // [A]
    }
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            printf("%4d", a[i][j]);
        printf("\n");
    }
    return 0;
}
```

Try to guess what the result will be.

3. In the above program, instead of [A] put in

   A: a[i][j]=i+1;
   B: a[i][j]=i+j+1;

   C: a[i][j]=i-j;

   D: a[i][j]=j-i;

   E: a[i][j]=(i+1)*(j+1);

Predict what the result will be.

6.3 The Eleventh Program

1. Execute the following program.

```
#include<stdio.h>
int main()
{
    int a[30][30]={0,};
    int i, j, k, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

    for(i=0;i<n;i++)
    {
        a[i][i]=i+1;    // [A]
    }

    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            printf("%4d", a[i][j]);
        printf("\n");
    }
    return 0;
}
```

2. In the above program, instead of [A] put in

   A: a[i][i]=n-i;
   B: a[0][i]=i+1;
   C: a[n-1][i]=i+1;
   라: a[i][0]=i+i;
   D: a[i][n-1]=i+1;
   E: a[i][n-i-1]=i+1;

Predict what the result will be.

6.4 The Twelvth Program

1. Execute the following program.

```
#include<stdio.h>
int main()
{
    int a[30][30]={0,};
    int i, j, k, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

    k=0;
    for(i=0;i<n;i++)
    {
        a[i][i]=k++;   // [A]
    }

    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            printf("%4d", a[i][j]);
        printf("\n");
    }
    return 0;
}
```

2. In the above program, instead of [A] put in

```
A: a[0][i]=k++;
B: a[n-1][i]=k++;
C: a[i][0]=k++;
D: a[i][n-1]=k++;
E: a[i][n-i-1]=k++;
F: a[n-i-1][i]=k++;
```

Predict what the result will be.

6.5 The Thirteenth Program

1. Execute the following program.

```
#include<stdio.h>
int main()
{
    int a[30][30]={0,};
    int i, j, k, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

    k=0;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
```

```
            a[i][j]=k++;    // [A]
    }

    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            printf("%4d", a[i][j]);
        printf("\n");
    }
    return 0;
}
```

2. In the above program, instead of [A] put in

```
    A: a[j][i]=k++;
    B: a[i][n-j-1]=k++;
    C: a[n-i-1][j]=k++;
    D: a[n-i-1][n-j-1]=k++;
    E: a[j][n-i-1]=k++;
    F: a[n-j-1][i]=k++;
```

Predict what the result will be.

6.6 The Fourteenth Program

1. Execute the following program.

```
#include<stdio.h>
int main()
{
    int i, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

    for(i=0;i<n;i++)
    {
        if((i%2)==0)
            printf("\ni=%d is even number", i);    // 짝수
        else
            printf("\ni=%d is odd number", i);     // 홀수
    }
    return 0;
}
```

For the first time, a program that uses the if statement. You need the if statement in two-dimensional array, so let’s tackle that now.

6.7 The Fifteenth Program

1. Execute the following program.

```
#include<stdio.h>
int main()
{
    int a[30][30];
    int i, j, n;
    printf("\nPlease enter number:");
    scanf("%d", &n);

    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            if(i==j)
                a[i][j]=9;
            else
                a[i][j]=1;
        }
    }

    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            printf("%4d", a[i][j]);
        printf("\n");
    }
    return 0;
}
```

The program outputs thus when number=5:

```
     9   1   1   1   1
     1   9   1   1   1
     1   1   9   1   1
     1   1   1   9   1
     1   1   1   1   9
```

A closer look shows only a[0][0], a[1][1], a[2][2], a[3][3], and a[4][4] have 9, while the others have 1. Why? Try figuring it out on your own. Asking yourself questions is a good way to start.

2. Execute the following program.

```
#include<stdio.h>
int main()
{
    int a[30][30];
    int i, j, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            if(i>j)
                a[i][j]=9;
            else
                a[i][j]=1;
        }
    }
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            printf("%4d", a[i][j]);
        printf("\n");
    }
    return 0;
}
```

The program outputs thus when number=5:

```
   1   1   1   1   1
   9   1   1   1   1
   9   9   1   1   1
   9   9   9   1   1
   9   9   9   9   1
```

Let’s find out why.

What will the output be when you substitute if(i<j) for if(i>j) ? What about if(i>=j) and if(i<=j) ?

Going back to the Fourteenth Program, the condition
    if((i%2)==0) is deduced.

    % is the remainder operator. i%2 means the remainder of i divided by 2. If i is even the remainder is 0. If i is odd the remainder is 1.
    Therefore the condition if((i%2)==0) is true when i is even.

Try to guess the output of the Fifteenth Program when if(i==j) is substituted by if((i%2)==0).

Now you are ready for two-dimensional array. All the tedious work is paying off. Two-dimensional array isn’t so bad, is it? Time to move on in the ‘curioser and curioser’ world of programming.

Chapter 7. Third Collection of Problems (Two-Dimensional Array)

Problems Set 3) In two-dimensional array, write a program that will output results according to the random numbers inputted.

3-1) Using a variable that increases by 1, write a program that fills the array.

```
    number = 5

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

3-2) Using a variable that increases by 1, write a program that fills the array.

```
    number = 5

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

3-3) Using a variable that increases by 1, write a program that fills the array.

```
   number = 5

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

3-4) Using a variable that increases by 1, write a program that fills the array. Make the program differ, progressing to the right when i is even, to the left when i is odd.

```
    number = 5

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

3-5) Write a program that fills outmost and diagonal lines in 9 and the rest in 0.

```
    number = 7

     9  9  9  9  9  9  9
     9  9  0  0  0  9  9
     9  0  9  0  9  0  9
     9  0  0  9  0  0  9
     9  0  9  0  9  0  9
     9  9  0  0  0  9  9
     9  9  9  9  9  9  9
```

3-6) Write a program that outputs 0 or 1 according whether the value of (i @ j) is even or odd. @ is one of the four operations (+,-,*,/).

```
    number = 5

     1  0  1  0  1
     0  1  0  1  0
     1  0  1  0  1
     0  1  0  1  0
     1  0  1  0  1
```

3-7) A twist on problem 3-6)

```
    number = 6

     1  1  0  0  1  1
     1  1  0  0  1  1
     0  0  1  1  0  0
     0  0  1  1  0  0
     1  1  0  0  1  1
     1  1  0  0  1  1
```

3-8) Another twist on problem 3-6)

```
    number = 6

     1  1  1  0  0  0
     1  1  1  0  0  0
     1  1  1  0  0  0
     0  0  0  1  1  1
     0  0  0  1  1  1
     0  0  0  1  1  1
```

3-6, 3-7, 3-8) are almost the same problems.

3-9) Using a variable that increases by 1, write a program that fills the array.

```
   number = 5

     1  2  3  4  5
    16  0  0  0  6
    15  0  0  0  7
    14  0  0  0  8
    13 12 11 10  9
```

3-10) Using a variable that increases by 1, write a program that fills the array.

```
   number = 5

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

3-11) Make it work only when the number is odd.

```
    number = 5

     3  3  3  3  3
     3  2  2  2  3
     3  2  1  2  3
     3  2  2  2  3
     3  3  3  3  3
```

3-12) Using a variable that increases by 1, write a program that fills the array.

```
    number = 5

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

3-13) Write an odd magic square program.

```
    number = 3

     8  1  6
     3  5  7
     4  9  2

    number = 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
```

Rules for making odd magic squares

1. Start in the middle of the first row in the array (3X3, or 5x5).
2. Aim diagonally right and up and put in the next number.

3. The line after the first is the last line.
4. The square next to the last is the first square.
5. When the square in the progressing direction is already filled,

go to the square beneath the original one.

Take a 3x3 odd magic square for example.

1. Start at a[0][1].
2. Go diagonally to the righthand top.
   The usual way is to a[-1][2] but as there is no -1 line go to a[2][2]
3. Go diagonally right and up.
   a[1][3] but there is no square for 3.
   Therefore you should go to a[1][0] .
4. Go diagonally right and up.
   The was is to a[0][1] but a number is already there.
   Go one square down, to a[2][0] .
5. Go diagonally right and up.
   Go to a[1][1] .
6. Go diagonally right and up.
   Go to a[0][2] .
7. Go diagonally right and up.
   Go to a[-1][3] . No −1 line or 3 square.
   Go to a[2][0] instead. A number is already there.
   Therefore go one square down from your original position, a[0][2] .
   Go to a[1][2] .
8. Go diagonally right and up.
   Go to a[0][3] . No square for 3.
   Go to a[0][0] .
9. Go diagonally right and up.
   Go to a[-1][1] . There is no -1 line.
   Therefore go to a[2][1] .
   The end.

Fill the magic square this way.

Chapter 8. No Longer a Novice: *Empire of the Ants* Sequence

You've mastered the magic square and nothing can stop you. However, the real challenge is yet to come. Conquering this obstacle will truly set you apart from the novices.

1. Take a look at this program.

```
#include<stdio.h>
int main()
{
    int i, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

    i=0;
    while(i<n) {
        printf("\ni=%4d", i);
        i++;
    }
    return 0;
}
```

The above program functions the same way as this one.

```
    for(i=0;i<n;i++)
        printf("\ni=%4d", i);
```

It not only executes in the same order, but the value of i after the for statement and the while statement is the same also.

The while statement decides whether to repeat or stop according to the conditions araised during execution. It is not merely a for statement substitute.

```
    while(hungry) {
        eat;
    }
```

The while statement above will stop eating when full. The amount eaten will differ by case.

2. Look at this program.

```
#include<stdio.h>
int main()
{
    int a[100]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, };
    int i, n;

    printf("\nPrint the contents of array a[]");

    i=0;
    while(a[i]>0) {
        printf("%4d", a[i]);
        i++;
    }
    return 0;
}
```

Among other things, I have put off explaining how to initialize an array. The one-dimensional array a[] given above initializes 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 0, 0, . . ., to 0 before the program starts.

Executing the above program will output thus:

```
1 2 3 4 5 6 7 8 9 10
```

Try to guess why.

3. Let’s look at this program.

```
#include<stdio.h>
int main()
{
    int a[100]={11, 12, 13, 14, 15, 6, 7, 8, 9, 10, };
    int i, n;

    printf("\nPrint the contents of array a[]");

    i=0;
    while(a[i]>0) {
        if(a[i]>10)
            printf("big ");
        else
            printf("small ");
        i++;
    }
    return 0;
}
```

The program outputs like this.

```
big big big big big small small small small small
```

Why?

4. I’m going to tinker with the program above to introduce the break statement.

```
#include<stdio.h>
int main()
{
    int a[100]={11, 12, 13, 14, 15, 6, 7, 8, 9, 10, };
    int i, n;

    printf("\nPrint the contents of array a[]");

    i=0;
    while(1) {
        if(a[i]<=0)
            break;
        if(a[i]>10)
            printf("big ");
        else
            printf("small ");
        i++;
    }
    return 0;
}
```

In while(1) the condition 1 is always true. Therefore the while statement above can go on forever. In such cases you can use the break statement to get out of the while statement. The program above executes exactly the same way as the preceding program, shown in 3. .

It is not customary for a break statement to appear so early in a while loop. Please note this is only an example.

5. Look at this program.

```
#include<stdio.h>
int main()
{
    int a[100]={11, 12, 13, 14, 15, 6, 7, 8, 9, 10, };
    int b[100]={0,};
    int i, n;
    i=0;
    while(a[i]>0) {    // program to copy array a[] to array b[]
        b[i]=a[i];
        i++;
    }
    printf("\nPrint the contents of array b[]");

    i=0;
    while(b[i]>0) {
        printf("%4d", b[i]);
        i++;
    }
    return 0;
}
```

The program above copies the contents of array a[] to array b[] .

We are now ready to face a programming task beyond novices. With all those examples in one-dimensional array, sharper students must have concluded the task will be about one-dimensional array.

The problem to solve is a sequence appearing in Bernard Werber’s novel *Empire of the Ants* (originally titled *Les Fourmis*). The sequence is given below.

```
        1
        1 1
        1 2
        1 1 2 1
        1 2 2 1 1 1
        1 1 2 2 1 3
        1 2 2 2 1 1 3 1
        1 1 2 3 1 2 3 1 1 1
        1 2 2 1 3 1 1 1 2 1 3 1 1 3
        1 1 2 2 1 1 3 1 1 3 2 1 1 1 3 1 1 2 3 1
        . . . . . . . . . . . . . . . . . . . .
        Progress to infinity.
```

From 1 in the first line comes 1 1 in the second line and from 1 1 comes 1 2 in the third line. Similarly the fourth line comes from the third, and so on. Those who have not read *Empire of the Ants* should try to guess the rules of the sequence. It is an interesting puzzle.

Write a program that runs the *Empire of the Ants* sequence, using two one-dimentional arrays. The program should output lines according to the amount of n inputted.

Use the while statement for the control structure in the *Empire of the Ants* sequence, not the break statement.

Chapter 9. Fourth Collection of Tasks(One-Dimensional Array)

You're running around bellowing you've cracked the Ants sequence.

Not to rain on your parade, try a few more problems before you leave apprenticeship forever.

1. Pascal's Triangle

```
                        1
                    1       1
                1       2       1
            1       3       3       1
        1       4       6       4       1
    1       5      10      10       5       1
.   .   .   .   .   .   .   .   .   .   .   .   .
```

Using two one-dimensional arrays, write a program that outputs Pascal's triangle.

2. Finding Primes With a Sieve

Using a 1000 sized one-dimensional array, write a program that finds all the primes up to 1000. Directions are given below.

Initialize the array from 0 to 999.

```
    for(i=0;i<1000;i++)
        a[i]=i;

    2 is a prime .
        Starting with 4, erase all multiples of 2 larger than 2.
        Putting 0 in a[] erases them.
        Erase 4, 6, 8, 10, 12, 14, ,,, etc.
        Find the next number to 2 that is not erased.
        3 is not erased.
    3 is a prime.
        Starting with 6, erase all multiples of 3 larger than 3.
        Erase 6, 9, 12, 15, 18, 21, ,,, etc.
        Find the next number to 3 that is not erased.
         5 is not erased.
    5 is a prime.
        Starting with 10, erase all multiples of 5 larger than 5.
        Erase 10, 15, 20, 25, 30, 35, ,,, etc.
        Find the next number to 5 that is not erased.
        7 is not erased.
    7 is a prime.
        Starting with 14, erase all multiples of 7 larger than 7.
        Erase 14, 21, 28, 35, 42, 27, ,,, etc.
        Find the next number to 7 that is not erased.
        11 is not erased.
    11 is a prime.
.....
```

Eliminating numbers in this order leaves the primes. Finally, print all the primes from 2 to 997.

Since multiples can be found using addition alone, write the program without using multiplication,.

3. Finding Primes With Prime Factor Analysis

Write a program to find 1000 primes using an array the size of 1000.

Directions are given below.

Since all primes except 2 are odd numbers, examine only the odds larger than 3. Odds that cannot be divided without remainder by a prime smaller than itself are prime.

For example, divide 17 by 3, 5, 7, 11, 13 to determine whether 17 is a prime. But thinking ahead, you do not need to try 5. If 17 can be divided without remainder by 5 you need to consider only when the quotient is larger than 5. If the quotient is smaller than 5 it should be 3, and this was already considered when you divided 17 by 3.

First put a0=2; a1=3; .

```
Starting with 5, examine all odds.
Check if 5 is divided cleanly by 3. Before dividing
    The square of 3 is 9. 9 is larger than 5.
    Therefore 5 is a prime. a[2]=5;
Check if 7 is divided cleanly by 3. Before dividing
    The square of 3 is 9. 9 is larger than 7.
    Therefore 7 is a prime. a[3]=7
Check if 9 is divided cleanly by 3. Before dividing
    The square of 3 is 9. 9 is the same as 9.
    Therefore 9 is not a prime.
Check if 11 is divided cleanly 3. Before dividing
    The square of 3 is 9. 9 is smaller that 11
    Check if 11 is divided cleanly by 3.
    It is not divided without remainder.
Check if 11 is divided cleanly by 5. Before dividing
    The square of 5 is 25. 25 is larger than 11.
    Therefore 11 is a prime. a[4]=11
Check if 13 is divided cleanly by 3. Before dividing
    The square of 3 is 9. 9 is smaller than 13.
    Check if 13 is divided cleanly by 3.
Check if 13 is divided cleanly by 5. Before dividing
    The square of 5 is 25. 25 is larger than 13.
    Therefore 13 is a prime. a[5]=13
Check if 15 is divided cleanly by 3. Before dividing
     The squre of 3 is 9. 9 is smaller than 15.
    15 is divided cleanly by 3. 15 is not a prime.
Check if 17 is divided cleanly by 3. Before dividing
    The square of 3 is 9. 9 is smaller than 17.
    17 is not divided cleanly by 3.
Check if 17 id divided cleanly by 5. Before dividing
    The squre of 5 is 25. 25 is larger than 17.
    Therefore 17 is a prime. a[6]=17
Check if 19 is divided cleanly by 3.
   . . . . .
```

In this way, write a program that outputs 1000 primes. Use the operator % to see if they are divided without remainder.

Chapter 10. Input and Output of Files

We have come a long, long way. Who knew it was so hard to cast away novitiate? But you are not truly ordained yet. Homework awaits you.

Here at the Programmer's Way, my goal was to combine the pupil's programming skills and understanding of programming languages in a way they would complement each other. Doing so I have put off some things you should have known but I thought best to put off till later. But those who have followed the Programmer's Way devoutly are already quite competent. I expect homework will not bother you at all.

1. Execute the following program.

```
#include<stdio.h>
int main()
{
    FILE *fin;
    char finname[80];
    int c;

    printf("\nPlease enter File name:");
    scanf("%s", finname);

    fin = fopen(finname, "r");

    c = fgetc(fin);
    while(c != EOF) {
        printf("%c", c);
        c = fgetc(fin);
    }
    fclose(fin);
    return 0;
}
```

When you execute the program it will tell you to input the File name. Give it any C program file name in the current directory.

Then this program will output that very C program onscreen.

Starting now, you have a great deal to learn how a program works. One concept is files. Let us say that files are an array of characters that exist outside a program. There is a very long array of characters. With files you can take one character at a time out of that array.

We can take characters into a program one by one, like this statement:

```
c = fgetc(fin);
```

Although the datum read is a character the datum received is an integer. There is a good reason that integers replace characters. **Do not receive data in the form of characters** hastily. The data type of c should not be changed to char.

Receiving input from the outside raises all sorts of situations. One situation is when you come to the end of a file. You have no characters to give the program that demands more. The special value EOF is needed in this case.

The value EOF must exceed the range of characters. That way it is distinguished from regular input characters. Therefore the form of C must be integers. Integers cover a wider range than characters.

One more point before we move on.

The structure shown in the program above, first reading a character and processing (outputting) the character read in the while statement then reading the character at the very bottem, is very important.

Let's take another look at it.

```
   c = fgetc(fin);
    while(c != EOF) {
        printf("%c", c);
        c = fgetc(fin);
    }
```

You must remember this control structure. Those who learn programming at the Programmer's Way will realise the significance of this control structure shortly. As it is a very common structure in programs please listen carefully, no matter how often it is mentioned.

==

File In/Output and Characters

A file is an array of characters outside a program. The characters we know are those we see onscreen. But characters in files include those that cannot be shown onscreen. Let us explore this more closely. The char form in C has a 1 byte size with the same range of values as integers from –128 to 127. When using integers without signs from 0 to 255 you must use the unsigned char form. The characters usually seen on a PC are ASCII code charaters. Let's take an example. The character A has no direct connection to the numeral 65. But inside a computer a byte with the value of 65 symbolizes A. In the same way B has the value of 66 and C has the value of 67. It is a promise we follow. We say that the ASCII code value of A is 65, the ASCII code value of B is 66 and the ASCII code value of C is 67.

Look at this example.

```
#include<stdio.h>
int main()
{
    unsigned char a, b, c;

    a = 'A';
    b = 'B';
    c = 'C';

    printf("\n%c %c %c ", a, b, c);
    return 0;
}
```

The output of this program is:

```
    A B C
```

    when changed to printf("\n%d %d %d ", a, b, c); the output is

```
    65 66 67
```

Another example.

```
#include<stdio.h>
int main()
{
    unsigned char a, b, c;

    a = 65;
    b = 66;
    c = 67;

    printf("\n%c %c %c ", a, b, c);
    return 0;
}
```

The output of this program is also A B C .

As seen in examples so far, characters are integers with range. Therefore they can be used in calculation or in the index of an array. Remember this carefully.

===

Including the example below, the input and output of files concerned here apply to Windows. Most file input and output work whether in Linux or Windows but there is a difference in line switch. Our work now is relevant to Windows text files.

1. Execute the following program.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    FILE *fin, *fout;
    char  finname[80];
    char  foutname[80];
    int   c;

    printf("\nPlease enter input file name:");
    scanf("%s", finname);
    printf("\nPlease enter output file name:");
    scanf("%s", foutname);

    fin = fopen(finname, "rb");
    if(fin==NULL) {
        printf("\nCannot open file:%s", finname);
        exit(1);
    }
    fout = fopen(foutname, "wb");
    if(fout==NULL) {
        printf("\nCannot open file:%s", foutname);
        exit(1);
    }

    c = fgetc(fin);
    while(c != EOF) {
        fputc(c, fout);
        c = fgetc(fin);
    }

    fclose(fin);
    fclose(fout);
    return 0;
}
```

Executing the program above sparks a request for the input file name. Give it any C program file in the directory, as you did in the previous program. This time the program requests an output file name. Give it another file name for output. You must give it a different name to preserve the files already there.

When executed properly, the program above will copy existing files to the new file name it received.

This program differs from the last example in a few ways. First, it uses two files. One as an input file and one as an output file.

Also, in the fopen spatement fin = fopen(finname, "rb"); the "rb" option is used. rb means input-only binary file attribute. The files that correspond to binary files are text files. Text files include text readable by the eye and control characters. Binary files are all files other than text files.

This is already getting complicated. Stay with me, please, even if it is hard to swallow.

The C programs we wrote so far were mostly text files, so they were usually opened with text files. However, text files can be opened with binary files. When opening with binary files, characters can be inputted and received one at a time, never missing even one. But opening with text files causes some characters contained in the original file to be missed, ignored. Two characters are used for line switch, but opening with text files sends us only one.
===

On Line Switch

```
    23 69 6E 63 6C 75 64 65   3C 73 74 64 69 6F 2E 68
    #  i  n  c  l  u  d  e    <  s  t  d  i  o  .  h
    3E 0D 0A 69 6E 74 20 6D   61 69 6E 28 29 0D 0A 7B
    >        i  n  t     m    a  i  n  (  )        {
    0D 0A 20 20 20 20 69 6E   74 20 69 2C 20 6E 3B 0D
                      i  n    t     i  ,     n  ;
    0A . . . . . . . . . . .
```

The hexadecimals above indicate each character when reading a file one character at a time, as shown below.

```
#include<stdio.h>
int main()
{
    int i, n;
. . . . . . . .
```

The characters 0D 0A can be seen after the first line switch <stdio.h>. Similarily 0D 0A are seen after main(). After int i, n; the characters 0D 0A are seen again. This is the line switch of text files when opening files with binary files.

But the character 0D is invisible when opening the file with a text file. Something blocks the character 0D from our view.

Try executing this program.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    FILE *fin;
    char  fname[80];
    int   c;

    printf("\nPlease enter input file name:");
    scanf("%s", fname);

    fin = fopen(fname, "rb");
    if(fin==NULL) {
        printf("\nCannot open file:%s", fname);
        exit(1);
    }

    c = fgetc(fin);
    while(c != EOF) {
        printf("%02X  ", c);
        c = fgetc(fin);
    }
    fclose(fin);
    return 0;
}
```

Give it a file that is not too long. That way the output can be seen in a single screen. Be sure to adjust two blank spaces in "%02X " so the output shows up tidily.

You can see that line switch changes when opening with text files, as in fin = fopen(fname, "r"); of the program above.

I have a reason for going incessantly on line switch, of course. Here are a few problems on file input and output.

Chapter 11. File Input and Output Problems

Problem 4-1) Write a program that reads files and and outputs the lines inverted, one at a time. (Use one-dimensional array to store them one line at a time and output them inverted)

```
    >h.oidts<edulcni#
    )(niam tni
    {
    ;n, j, i tni
    . . . . . .
```

Problem 4-2) Write a program that reads files and outputs in the form below. (Use two-dimensional array to store, then output all at once)

```
   . . . . . . . .
   . . . . . . . .
   . . . . . . . .
       int i, j, n;
   {
   int main()
   #include<stdio.h>
```

Problem 4-3) Write a program that reads files and outputs in the form below. (Use two-dimensional array to store, then output all at once)

```
     #  i  {         . . . . . . . .
     i  n            . . . . . . . .
     n  t            . . . . . . . .
     c               . . . . . . . .
     l  m      i     . . . . . . . .
     u  a      n     . . . . . . . .
     d  i      t     . . . . . . . .
     e  n            . . . . . . . .
     <  (      i     . . . . . . . .
     s  )      ,     . . . . . . . .
     t               . . . . . . . .
     d         j     . . . . . . . .
     i         ,     . . . . . . . .
     o               . . . . . . . .
     .         n     . . . . . . . .
     h         ;     . . . . . . . .
     >               . . . . . . . .
     . . . . . . . . . . . . . . . .
```

Problem 4-4) Write a program that reads two files and outputs them as one. Combine them by line, two lines to one. Put in $$$ between the lines of each file to distinguish one file from the other. (Use one-dimensional array to store one line at a time, then output)

```
    #include<stdio.h>$$$#include<stdio.h>
    int main()$$$int main()
    {$$${
    int i, j, k;$$$int a[10]={0,};
    . . . . . . . . .
    . . . . . . . . .
```

Problem 4-5) Write a program that outputs inversely the column that is the second file in Question 4-4.

```
    #include<stido.h>$$$>h.oidts<edulcni#
    int main()$$$)(niam tni
    {$$${
    int i, j, k;$$$;},0{=]01[a tni
    . . . . . . . . .
    . . . . . . . . .
```

Question 4-6) Write a program that converts a random file to hexadecimal. (Use one one-dimensional array the size of 16 and one input file to solve the task.) Make it output thus:

1. The first six places mark relative location.
2. Then output 16 characters at a time.
3. Print hexadecimals first and ASCII characters next.
4. Print eight hexadecimals and leave one more blank.
5. Leave three spaces between hexadecimals and ASCII characters.
6. Mark unprintable characters with '.'.
   (Use the isprint function to see if the character is printable)
7. Output the last line as shown below.

```
000000 23 69 6E 63 6C 75 64 65  3C 73 74 64 69 6F 2E 68   #include<stdio.h
000010 3E 0D 0A 69 6E 74 20 6D  61 69 6E 28 29 0D 0A 7B   >..int main()..{
000020 0D 0A 20 20 20 69 6E 74  20 69 2C 6A 2C 6E 2C 6B   ..   int i,j,n,k
000030 2C 6C 3B 0D 0A 20 20 20  69 6E 74 20 61 5B 33 30   ,l;..   int a[30
000040 5D 5B 33 30 5D 3D 7B 30  2C 7D 3B 0D 0A 0D 0A 20   ][30]={0,};....
000050 20 20 70 72 69 6E 74 66  28 22 6E 75 6D 62 65 72   printf("number
000060 3D 22 29 3B 0D 0A 0D 0A  20 20 20 73 63 61 6E 66   =");....   scanf
000070 28 22 25 64 22 2C 26 6E  29 3B 0D 0A 0D 0A 20 20   ("%d",&n);....
000080 20 66 6F 72 28 69 3D 30  3B 69 3C 6E 3B 69 2B 2B    for(i=0;i<n;i++
000090 29 0D 0A 20 20 20 7B 0D  0A 20 20 20 20 20 20 66   )..   {..      f
0000A0 6F 72 28 6A 3D 30 3B 6A  3C 6E 3B 6A 2B 2B 29 0D   or(j=0;j<n;j++).
0000B0 0A 20 20 20 20 20 20 7B  0D 0A 09 09 61 5B 69 5D   .      {....a[i]
          .     .    .    .       .    .    .    .
000230 20 20 20 7D 0D 0A 0D 0A  20 20 20 72 65 74 75 72      }....   retur
000240 6E 20 30 3B 0D 0A 7D 0D  0A                         n 0;..}..
```

The program below will help you with inputting and outputting files.

```
#include<stdio.h>
int main()
{
    FILE *fin;
    char fname[80];
    unsigned char line[100];
    int  c;

    printf("\nPlease enter file name:");
    scanf("%s", fname);

    fin = fopen(fname, "rb");
    if(fin == NULL) {
        printf("\nCannot open file: %s", fname);
        exit(1);
    }

    c = fgetc(fin);
    while(c ! = EOF) {
        if(c != 13) {
            Put in array
        }
        else {
             13. In hexadecimal, a 0D character.
             You must discard the 0D character to process line switch.
             Similarily 0A character must also be read and discarded.
             Ready to read the next line.
             Do what must be done at the end of the line.
        }
        c = fgetc(fin);
    }

    You have reached the end of the file.
    Sometimes it ends without line switch.
    In this case the stored line slipped unprocessed out of the while
statement. It must be finished.

    This occurs not only in file input and output but in many cases,
so this control structure comes in handy.

    fclose(fin);
    return 0;
}
    To summarize

    c = fgetc(fin);
    while(c != EOF) {
        process the character read.
        c = fgetc(fin);
    }
    Follow up.
    The read files are processed thus:
```

```
    if(c != 13) {
        store in array.
    }
    else {
        met 13 (hexadecimal character 0D). Must process the following
character 0A. Then the start of the next line can be read.
        Output characters stored in the array.
    }
```

Sometimes two files are read.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    FILE *fin1, fin2;
    char fname1[80], fname2[80];
    unsigned char line[100];
    int  c, d;

    printf("\nPlease enter file name:");
    scanf("%s", fname1);
    printf("\nPlease enter file name:");
    scanf("%s", fname2);

    fin1 = fopen(fname1, "rb");
    if(fin1 == NULL) {
        printf("\nCannot open file: %s", fname1);
        exit(1);
    }

    fin2 = fopen(fname1, "rb");
    if(fin2 == NULL) {
        printf("\nCannot open file: %s", fname2);
        exit(1);
    }

    c = 0;
    d = 0;
    while((c != EOF) && (d != EOF)) {
        c = fgetc(fin1);
        while((c ! = EOF)&&(c != 13)) {
            put in array
            c = fgetc(fin1);
        }
        // Slipped out of while loop. Encountered EOF or 13.
        if(c != EOF) {
             13. 0D character in hexadecimal.
             Discard 0D character to process line switch.
             Read and discard 0A character.
             Ready to read next line.
             Do what must be done at the end of the line.
        }
        else {
            // If there is a stored line do what must be done at the
end of the line.
        }
        d = fgetc(fin2);
        while((2 ! = EOF)&&(2 != 13)) {
            put in array
            d = fgetc(fin2);
        }
        // Slipped out of while loop. Encountered EOF or 13.
        if(c != EOF) {
             13. 0D character in hexadecimal
             Discard 0D character to process line switch.
             Read and discard 0A character.
             Ready to read next line.
             Do what must be done at the end of the line.
        }
        else {
            // If there is a stored line do what must be done at the
end of the line.        }

    }

    Reached the end of the file. The end of file1 or file2.

    Process remaining files. The structure above will be repeated
almost identically.

    fclose(fin1);
    fclose(fin2);
    return 0;
}
```

Chapter 12. Functions

The second obstacle I have put off is functions.

In the programs so far, we did not write functions, using only main functions to write programs. But the need to write functions arises when you want to split a program in function-sized bits. I believe you are at that stage.

You appreciate something when you need it most, so I have put off functions until now.

Functions can be written the same way we wrote main functions. The only difference is the main functions we wrote had no factors. Functions written by the user usually need factors and it is important to understand this properly.

Look at this example.

```
#include<stdio.h>
// place to define oneline function
int oneline(int n)
{
    int i;

    for(i=0;i<n;i++)
        printf("%4d", i+1);
    return 0;
}

int main()
{
    int i, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

    for(i=0;i<n;i++) {
        oneline(i+1);   // use oneline function
        printf("\n");
    }
    return 0;
}
```

The program above does the same work as the program below.

```
#include<stdio.h>
int main()
{
    int i, j, n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

    for(i=0;i<n;i++) {
        for(j=0;j<i+1;j++)
            printf("%4d", j+1);
        printf("\n");
    }
    return 0;
}
```

In the program above the one line function is not used as customary, just to show what the function is.

Like the equation y = f(x) in math, functions are used to produce the result y from the given x. It is easy to think that only the results count and the process to the result is unimportant. But as seen in the program above, the result 0 from the function oneline is insignificant while the numbers printed onscreen matter.

Therefore in C language not only the final result of a function but effects during the process may play an important part.

Look at the following program.

```
#include<stdio.h>

int factorial(int n)
{
    int i, fact;

    fact = 1;
    for(i=2;i<=n;i++)
        fact = i*fact;
    return fact;
}

int square(int n)
{
    return n*n;
}

int main()
{
    int n;

    printf("\nPlease enter number:");
    scanf("%d", &n);

    printf("\nnumber=%4d factorial = %4d square = %4d ",
        n, factorial(n), square(n));
    return 0;
}
```

We used the functions factorial and square in the program above. In this case the result value matters.

Chapter 13. Final Collection of Problems

With functions we have completed the beginner’s course. Congratulations to all, and now the intermediate course. Before we move on, a few more difficult tasks for our fledgling graduates.

```
       #
  #### #
     # #
     # ##
     # #
       #
```

1. Write a Program that outputs above character. (put necessary information in 2dimensional array)

2. This time put necessary information in size 6 one-dimensional array. You need 7 position of existence of #. You can do this since one character has 8 bits.

3 Write a program that scrolls above character to left ans right. Use left arrow and right arrow key. Left scroll means

```
1 2 3 4 5   becomes

2 3 4 5 1   and

3 4 5 1 2   becomes.
```

4. Write a program that rotates the characters '\', '|', '/' '-' stored in the one-dimensional array.

5. Write a program that controls speed and rotation direction in the above task. Use the up arrow key to increase speed, the down arrow key to decrease, and the right and left arrow keys to change direction.

6. Write a calender program.

7. Write a three digit baseball game program.

8. Write a pop-up menu program.

9. Write a ladder game program.

10. Write an Othello game program.

11. Write a Minesweeper game program.

12. Write a PushPush game program.

Chapter 14. Following Up: C/C++, Data Structure

You are more confident now you know the basics of programming. But you still do not know what and how you must learn, what goal to study for.

First you must acquaint yourself with inputting and outputting files. Then you must learn sorting algorithms and try to implement them. Although sorting algorithms are usually taught at the beginner’s stage, the author thinks materializing sorting algorithms is rather beyond beginners. But those past the beginner’s stage, especially those who passed at Programmer’s Way, should have no difficulty materializing sorting algorithms.

The bubble sort, selection sort and insertion sort are relatively easy to understand and implement. The exception is the quick sort. Since the quick sort uses recursive call you will be in trouble unless you become used to recursive functions. Use paper cards to solve the Hanoi Tower problem and implement it in a program.

Let’s go over what you should study. There are many things you must know. It is time for structure and pointer, which we glossed over during C language. You must know structure and pointer to understand data structure. The ability to implement data structure is essential for complicated operations in programs.

Teach yourself data structure properly with books like *Learning Algorithm with C* (Lee, Jaegyu). Learn about stacks, queues and linked lists and try to implement them. Use arrays to implement linear search and binary search, then learn trees to implement a binary search tree. Study and implement balanced trees(2-3 tree, 2-3-4 tree, red-black tree, B-tree). After that, study hashing. Since hashing is very important, you must not only understand but implement. Those who have the time will benefit from studying the heap sort, too.

That about concludes the intermediate course. Those who have finished the intermediate course are ready to implement Dad’s Puzzles among the challenge problems.

Chapter 15. Now What? : Choosing Your Path

Now you know sorting and data structure, now you have implemented them, it is time for you to face the real world. Time to write real, usable programs. That is, you are going to be a professional programmer.

There are many ways to professional programming.

You could stick to Visual Basic.

You could use Windows API and C/C++. Or you could use MFC.

You could use Java. C# is the Microsoft way.

And so on, with different things to learn whether you choose web programming, DB programming, network, hardware control or mobile, etc.

Above all you need object-oriented training. According to my experience and current trends, you must be accustomed to an object-oriented develop environment. Although some people are fine with only Visual Basic or Power Builder, the norm is OOP training. OOP develop environment has roughly three branchs, C++, Java, and C#. There is also Delphi and other object-oriented languages.

The writer thinks Java is the best way among the three. The training mentioned to acquire intermediate skills, file input and output, sorting, structure-pointer, linked lists, trees, balanced trees and hashing, should be easier with Java, and OOP training as well.

Of course, intermediates can start with Java, too. Since most training in Java is effective in C++ and C#, you can worry which path to choose after you have improved your skills.

Chapter 16. Polish Your Skills

I have the sinking feeling that answers or sources to the tasks at Programmer's Way will haunt the Internet.

What will you do? The task is stubborn. You've been on the problem for a week. You're annoyed, frusturated, downright mad. You want the answer. You really, really want to peek.

No. I will not. I shall not. I'm doing this on my own.

Your attitude in such crises will change who you become.

The choice is yours. Peek or not.

Be a programmer or not.

And your future…