Tutorial 4- Week 4

Objectives: To practice with
- for, while, do…while repetition statements

1. What is displayed by this program fragment for an input of 8?

```
scanf("%d", &n);
ev = 0;
while (ev < n) {
    printf("%3d", ev);
    ev = ev + 2;
}
printf("\n");
```

Answer:

Output: `0  2  4  6`

2. Write a program fragment that produces this output:

```
0   1
1   2
2   4
3   8
4  16
5  32
6  64
```

Answer:
Since we know the number of iterations (7), it is safer to user a for loop instead of we\hile as follows:

```
    int i, value;

   for(i=0; i<=6;i++){
    value= pow(2,i);
    printf("%3d %4d\n", i, value);
    }
}
```

We can also use a while loop as follows

```
    i = 0;
    while (i <=6)
    {
       value = pow (2, i);
       printf("%3d %4d\n", i, value);
       i = i + 1;
    }
```

You can also use a Do..While since the loop has to execute at least once.

```
i=0;
do
{
value= pow(2,i);
printf("%3d %4d\n", i, value);
i++;
}
while(i<=6);
```

To use the function pow, the header math.h should be declared.

3. What errors do you see in the following fragment? Correct the code so it displays all multiples of 4 from 0 through 100.

```
for mult4 = 0;
mult4 < 100;
mult4 += 4;
printf("%d\n", mult4);
```

Answer:

```
for (mult4 = 0;  mult4 <= 100;  mult4 += 4)
    printf("%d\n", mult4);
```

4. Show the output displayed by these nested loops:

```
for (i = 0; i < 3; ++i) {
  printf("Outer %4d\n", i);
  for (j = 0; j < 2; ++j) {
       printf(" Inner%3d%3d\n", i, j);
    }
  for (k = 2; k > 0; --k) {
       printf(" Inner%3d%3d\n", i, k);
    }
}
```

Answer:

```
Outer    0
   Inner   0  0
   Inner   0  1
   Inner   0  2
   Inner   0  1
Outer    1
   Inner   1  0
   Inner   1  1
   Inner   1  2
   Inner   1  1
Outer    2
```

```
Inner   2  0
Inner   2  1
Inner   2  2
Inner   2  1
```

5. Write nests of loops that cause the following output to be displayed:

```
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
0 1 2 3 4
0 1 2 3
0 1 2
0 1
0
```

Answer:

Given the relative complexity of the task, design first a solution to the problem before starting your code

The pattern has two sides

```
0
0 1          Increasing depth
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
0 1 2 3 4
0 1 2 3
0 1 2         Decreasing depth
0 1
0
```

If we take the rightmost edge (edge)5 as a reference, we can note that:

- The top side has "edge+1" lines. In each line we print a sequence of integer values. If we generate this side content using a loop, with iterator i that's incremented in every iteration, we can note that the sequence to print in every line is [0:i]

```
Iterator i       values
         0         0
         1         0 1
         2         0 1 2
         3         0 1 2 3
         4         0 1 2 3 4
         5         0 1 2 3 4 5    where 5 corresponds to edge
```

- The bottom side has "edge" lines. The depth of each line decreases at every iteration. If we generate this side's content using a loop, with iterator i that's decremented in every iteration, we can note that the sequence to print in every iteration is [0:i]:

```
Iterator i   value
         4   0 1 2 3 4   , where 4 corresponds to "edge-1"
         3   0 1 2 3
         2   0 1 2
         1   0 1
         0   0
```

Given our design solution above we can deduce the following code:

```
#include<stdio.h>

#define EDGE 5

int main() {
int i,j;
for (i=0; i<=EDGE;i++)
{
   for (j=0; j<=i;j++)
      printf("%d ", j);
    printf("\n");
}
for (i=EDGE-1; i>=0;i--)
{
   for (j=0; j<=i;j++)
      printf("%d ", j);
    puts("");
}
return 0;
}
```

Try it with different EDGE value in the define directive.

IMPORTANT: please note that we need to add { after the first outer loop; otherwise printf("\n) or puts will be out of the outer loop's scope; try it to see the effect.

6. Rewrite the following code using a do-while statement with no decisions in the loop body:

```
sum = 0;
for (odd = 1; odd < n; odd = odd + 2)
   sum = sum + odd;
printf("Sum of the positive odd numbers less than %d is
      %d\n", n, sum);
```

```
 sum = 0;
odd = 1;
do {
   sum = sum+odd;
   odd = odd+2;

} while (odd < n);

printf("Sum of the positive odd numbers less than %d is
%d\n", n, sum);
```

In what situations will the rewritten code print an incorrect sum?

With the do..while, the loop body executes at least once. Therefore, the code will print an incorrect sum, when the loop in the original code should not have been executed in the first place, i.e. when the for condition is not true at the start. This corresponds to "1<n" being false, i.e. 1>=n

Therefore, an incorrect result is displayed when n<=1.

7. **Design an interactive input loop that scans pairs of integers until it reaches a pair in which the first integer evenly divides the second.**

Answer:

You 'll have to write a program to prompt the user to continuously input two numbers, until the first number evenly divides the second; i.e. the first number is a divisor of the second (modulo % operator can be used)

```
int num1, num2;

do {
   /* Get two numbers. */
   printf("Enter a pair of integers separated by space>");
   scanf("%d%d", &num1, &num2);
} while (num2 % num1 != 0);
```

8. **What does the following code segment display? Try each of these inputs: 345, 82, 6. Then, describe the action of the code.**

```
printf ("\n Enter a positive integer> ");
scanf("%d", &num);
do {
        printf("%d ", num % 10);
        num /= 10;
} while (num > 0);
printf("\n");
```

Answer:

```
Enter a positive integer> 345 5 4 3
Enter a positive integer> 82 2 8
Enter a positive integer> 6 6
```

The code displays the digits of an integer in **reverse order** and separated by spaces.

9. Write a do-while loop that repeatedly prompts for and takes input until a value in the range 0 through 15 inclusive is input. Include code that prevents the loop from executing forever on input of a wrong data type.

Answer:

If we did not have to verify that the read data is of its specified data type, the following code will be sufficient:

```
#include<stdio.h>
#include<stdbool.h>

int main(void)
{

int data;
bool repeat= false;
do
{
if(repeat)
printf("\n invalid value, your input should be integer in the range [0:15]: ");
else {
 printf("\n please input an integer value in [0:15]: ");
  repeat=true;
 }
scanf("%d", &data);

} while (data <0 || data> 15);
printf(" Valid Data: %d", data);
}
```

The above code works fine, if the input data is always an integer. If it is not, and the local variable data is not initialized to zero (being a local variable, there is no guarantee its value would be initialized to zero), the above code leads to infinite loop (try it!) as scanf will keep trying to read the data from the same (non-empty) input buffer.

To ascertain that the read value by `scanf` was indeed an integer, we need to check the return value of `scanf` as follows:

status=scanf("%d", &data);

The value of *status* will be 1 if scanf read one integer value as instructed by the format specifier %d. This is valid for any format specifier regardless even of their numbers in the scanf argument.

If the input value is not integer (i.e. status!=1), we need to clear the buffer first from the user input before reading a new , hopefully valid, data from the user. To clear the keyboard buffer, we have to read its content. To this end, we suggest three solutions in order to read one line input from the user up to the new line character.

For the first two solutions, we first need to define an array of characters to store the input's line:

```
#define LINE_SIZE 100
char line[LINE_SIZE];
```

The first two solutions make use of

1 The built-in function gets(…) ; although the built-in function fgets is more efficient(more when we study files)
2 The regular expression [^\n] to read all characters up to new line: scanf("%[^\n]", line)

The third solution consists of skipping all the characters until the "new line" character using a "do..while" with scanf to a single character.

The function definitions of the above solutions are:

```
void skip_line_gets(){
char line [LINE_SIZE];
gets(line);
return;
}


void skip_line_format(){
char line [LINE_SIZE];
scanf("%[^\n]",line);
return;
}

void skip_line_char(){
char ch;
do
scanf("%c",&ch);

while(ch!='\n');
return;
}
```

Because line is an array, we don't need to prefix it with "&" in the scanf
IMPORTANT: gets function is deprecated, use instead the functon fgets, see Lect 5B

The resulting program is:

```
#include<stdio.h>
#include <stdbool.h>
#define LINE_SIZE 100
void skip_line_gets();
void skip_line_format();
void skip_line_char();


int main(void){
        bool inValid=true;  // while loop conditional expression
        int status; // return value of scanf
```

```
        int data;
                /* Get data from user until in_val is in the range 0   15. */
        do {
                /* Get a number from the user. */
                printf("Enter an integer in the range [0:15]>: ");
                status = scanf("%d", &data);
                /* Validate the number. */
                if (status != 1) {  /* an integer value was NOT read */
                   //skip_line_gets();
                   // skip_line_format();
                   skip_line_char();
                   printf("InValid Input \n");
                }

                else if (data < 0 || data > 15) {  /* data is valid but out of range */
                        printf("Number %d is not in requested range.\n", data);
                }

                else
                   {
                        inValid=false;   /* to escale the loop */
                        printf ("Valid Input \n");
                }
        } while (inValid);




}


void skip_line_gets(){
char line [LINE_SIZE];
gets(line);
return;
}


void skip_line_format(){
char line [LINE_SIZE];
scanf("%[^\n]",line);
return;
}

void skip_line_char(){
char ch;
do
scanf("%c",&ch);
```

```
while(ch!='\n');
return;
}
```

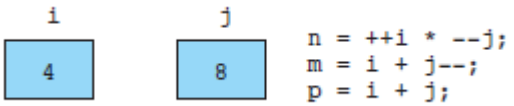
10. Where possible, write equivalents for the following statements using compound assignment operators:

```
a. s = s / 5;
b. q = q * n + 4;
c. z = z - x * y;
d. t = t + (u % v);
```

Answer:

```
a. s /= 5;
b. Not possible since (q * n) cannot be separated, and (n + 4)
   is not parenthesized.
c. z -= x * y;
d. t += u % v;
```

11. What values are assigned to n, m, and p, given these initial values?



Answer:

1. For the first statement, both i and j are updated before calculating the value of n. As such, i= 5, j=7, and n= 5*7=35
2. The current value of j is first used to calculate "i+j"; therefore m=12; then j is decremented, i.e. j=6
3. P=5+6=11

```
n = 35; m = 12; p = 11;
```