

## S&T-1st C Programming Lab 2

### Iterative Instructions

31 January 2025 Friday 6:45pm

---

#### Assignment 1

Unzip Lab2.zip sent to you by email attachment.

A *loop* can be constructed using `for`, `while` or `do while` iterative instructions (here we only use `for` instruction). A *nested-loop* contains more than one iterative instructions fitting one within another as shown in ex3.c. The program is used to animate a flying arrow.

```
// ex3.c
// Example used to animate a flying arrow. Lab2

#include <stdio.h>
#include <stdlib.h> // for clear screen
#include <windows.h> // for Sleep()

main()
{
    int t, j;

    for (t=0; t <25; t++)
    {
        system ("cls"); // clear screen

        for (j=0;j<=2*t;j++) printf (" ");

        printf ("--->");
        Sleep(250); // wait 0.25 seconds
    }

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

In the above program <stdlib.h> contains a useful function `system ("cls")` which can be used to clear the contents on the screen. This function will be used in our Assignment 2. Another library is the <windows.h> which contains the function `Sleep()`. Its resolution is specified in millisecond. Therefore `Sleep(250)` will pause for 250 millisecond (or 0.25 second).

The nested loop is used to print spaces. The number of spaces printed depends on the values of `t` as shown in the following table.

t	no. of spaces printed
0	1
1	3
2	5
3	7
4	9
:	:

Once the spaces are printed, an arrow will be displayed on the screen. After a pause of 0.25 second to allow the viewer to register the position of the arrow, the next iteration begins by cleaning the screen and performs every thing all over again. The visual effect of this program is the animation of a flying arrow. Please compile [ex3.c](#) and run it.

In [ex4.c](#), line 8 prints 13 lines to position the inverted "V" at appropriate altitude. In the nested for loop, line 11 prints one more blank line. Line 12 displaces 17 columns. The inverted "V" is printed by lines 13 to 16. Compile and run this program.

```

1:  // ex4.c
2:  // To print an inverted V on 4 rows and at the lower part of screen. Lab2.

3:  # include <stdio.h>

4:  main()
5:  {
6:      int i,j,t;

7:      t=8;

8:      for (i=0;i<21-t;i++) printf ("\n");

9:      for (i=0; i<4; i++)
10:     {
11:         printf ("\n");
12:         for (j=0;j<=2*t;j++) printf (" ");

13:         for (j=0;j< 3-i;j++) printf (" ");
14:         printf ("*");

15:         for (j=0; j <2*i; j++) printf (" ");
16:         printf ("*");
17:     }

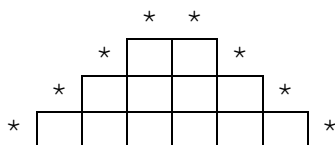
18:     printf ("\n\n");
19:     return 0;
20: }

```

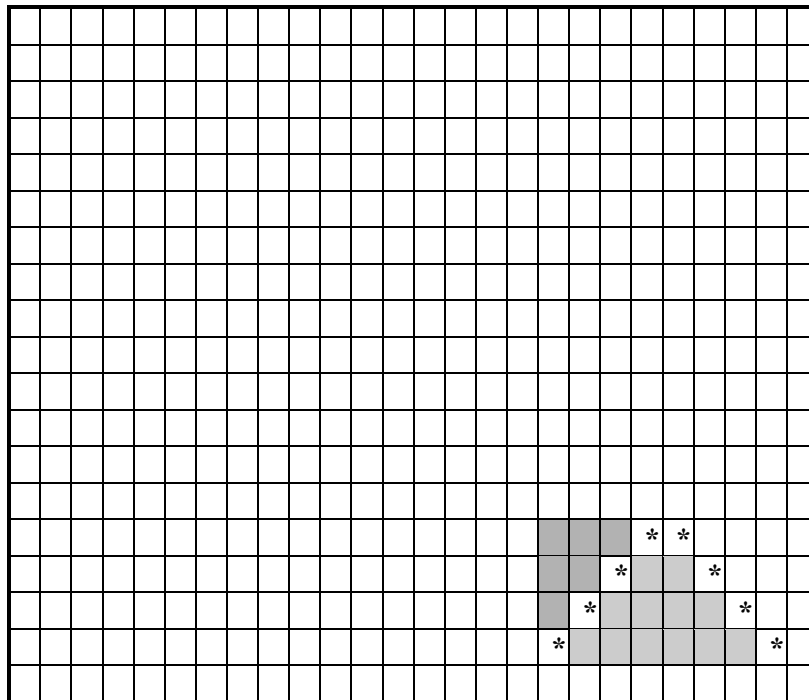
The indices used in [ex4.c](#) are illustrated in the following two tables.

i	no. of spaces 3-i			
i=0	3			
i=1	2			
i=2	1			

i	no. of spaces 2*i					
i=0	0					
i=1	2					
i=2	4					
i=3	6					

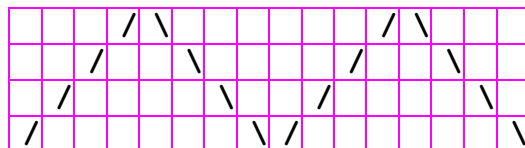


The screen display of **ex4.c** is as follows:

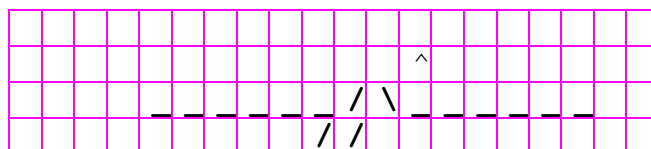


In this assignment we are going to write a program named as **bird.c** to animate a bird flying from the left bottom corner to the right top corner of the monitor screen. To animate the bird a simple technique is to draw the bird, erase it and draw it again in a position slightly further away. Doing this rapidly and continually fools the eyes into seeing a flying bird. We first construct two basic patterns used in the animation.

i. Fold Position



ii. Straight Position



The following algorithm may be used in your program:

```

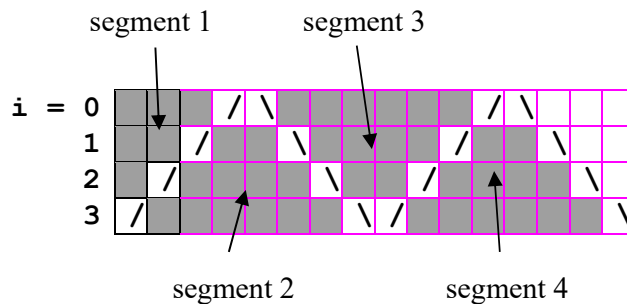
loop varying t from 0 to 21
{
  clean the screen;
  position the bird at an appropriate altitude based on t;
  displace the bird based on t, and display fold pattern (nested loop should be used);
  delay 0.5 second;
  clean the screen;
  position the bird at an appropriate altitude based on t;
  displace the bird based on t, and display straight pattern;
  delay 0.5 second;
}

```

## Help

You can execute the **bird\_demo.exe** to view the animation.

We first analyze the 4 segments in the fold pattern. Let  $i$  be the row index, and  $j$  the column index and  $j$  is reset in each segment.



**Segment 1:** Fill in the blanks

i	num of spaces (j)
0	
1	
2	
3	
i	j=

**Segment 2:** Fill in the blanks

i	num of spaces (j)
0	
1	
2	
3	
i	j=

### Segment 3: Fill in the blanks

i	num of spaces (j)
0	
1	
2	
3	
i	j=

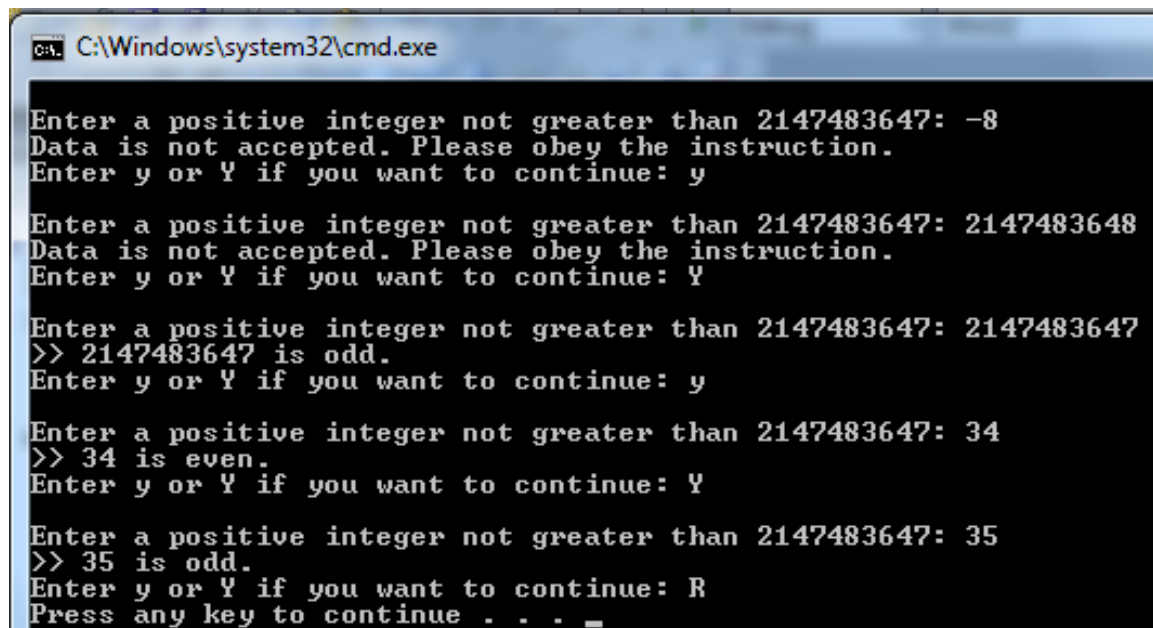
Segment 4 is same as segment 2.

To print a / use `printf ("/")`. Use `printf ("\\")` to print a \. Why?

You may hard code the straight pattern. Be aware of the \ in \_\_\_\_/\_\_\_\_.

### Assignment 2

Write a program named as check.c to inform the user to enter an integer from the keyboard, and the program will check whether the integer is even or odd as shown on the screen. The program will **repeatedly** inform the user to indicate to continue or not, and will execute accordingly. A complete session of the execution is shown as follows:



```
C:\Windows\system32\cmd.exe
Enter a positive integer not greater than 2147483647: -8
Data is not accepted. Please obey the instruction.
Enter y or Y if you want to continue: y

Enter a positive integer not greater than 2147483647: 2147483648
Data is not accepted. Please obey the instruction.
Enter y or Y if you want to continue: Y

Enter a positive integer not greater than 2147483647: 2147483647
>> 2147483647 is odd.
Enter y or Y if you want to continue: y

Enter a positive integer not greater than 2147483647: 34
>> 34 is even.
Enter y or Y if you want to continue: Y

Enter a positive integer not greater than 2147483647: 35
>> 35 is odd.
Enter y or Y if you want to continue: R
Press any key to continue . . . _
```

### Assignment 3

Name your program as **prime.c** and it is organized as follows:

- (i) Write a function that accepts an int argument called *this1* and returns **1** if *this1* is prime. Otherwise the function returns **0**.

The function header is given below.

**int check\_prime (int *this1*)**

To check if *this1* is prime, we only have to check the factors from 2,3,4,5,6,7,8,.. up to  $\frac{this1}{2}$  (inclusive). You can use a for loop.

- (ii) Write a main function that asks the user to enter an integer, and uses the `check_prime` function to check if the number entered is prime. Print the output on the screen.

#### Test Run 1:

Enter an integer larger than 2 and less than 2147483647 : 23  
23 is a prime number.

#### Test Run 2:

Enter an integer larger than 2 and less than 2147483647 : 24  
24 is not a prime number.

**The programming lab is only 2 hours.  
Please prepare your programs in  
advance.**