Lecture 4

Pointers to: pointers; arrays and structures.

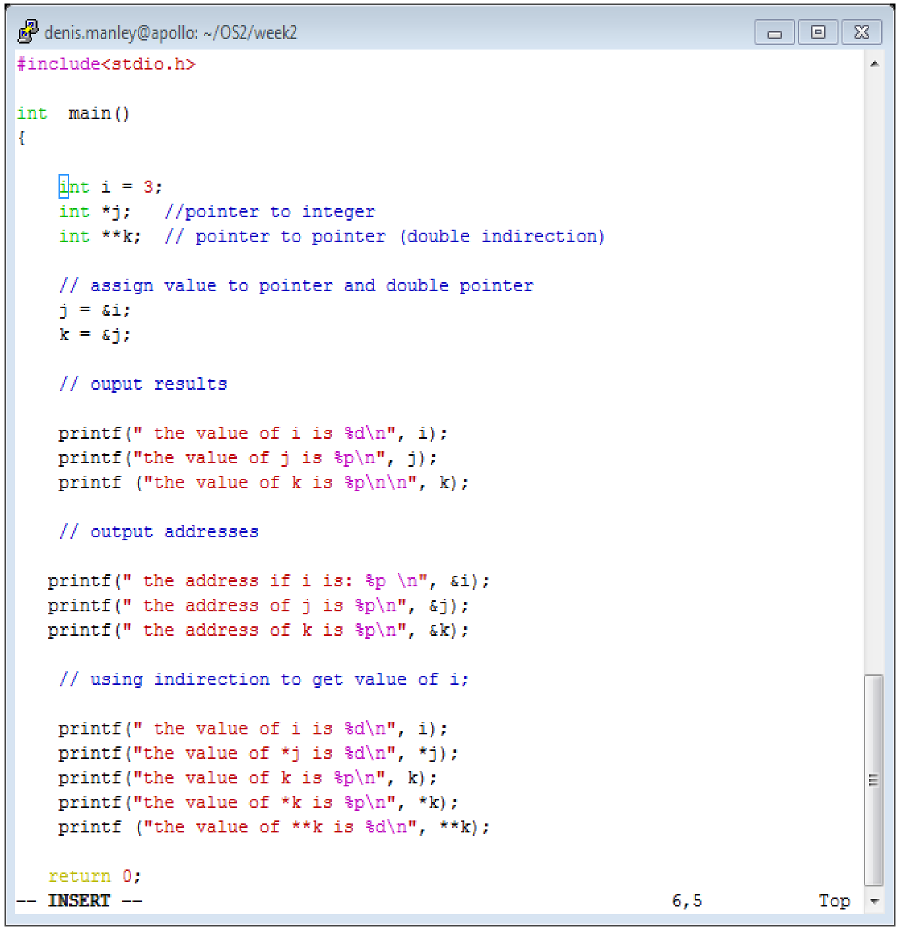
Pointers to pointers

- A pointer can also pointer to another pointer which in turn pointers to a “standard” variable:
  - **int i=3; // an integer variable**
    int *j; // a pointer
    int **k; // a pointer to a pointer (double pointer)
  - **j=&i; //line 1 (assigned address of an integer)**
  - **k=&j; //line 2 (assigned the address of a pointer)**

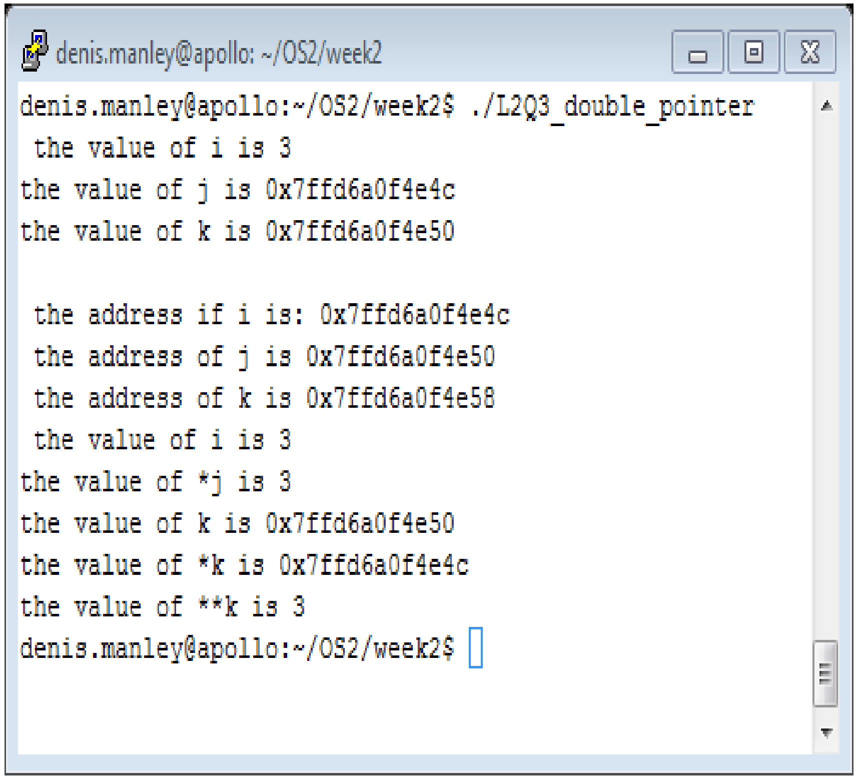
Examples

- **What is output of the following statements**
  - **printf(“%d”, **k);**
  - **printf(“%p”, *k);**
    printf(“%d”,*j);
    printf(“%d”,i);
  - **Assume the following: i = 3, j = &I; k = &j**

Double_pointer code and output



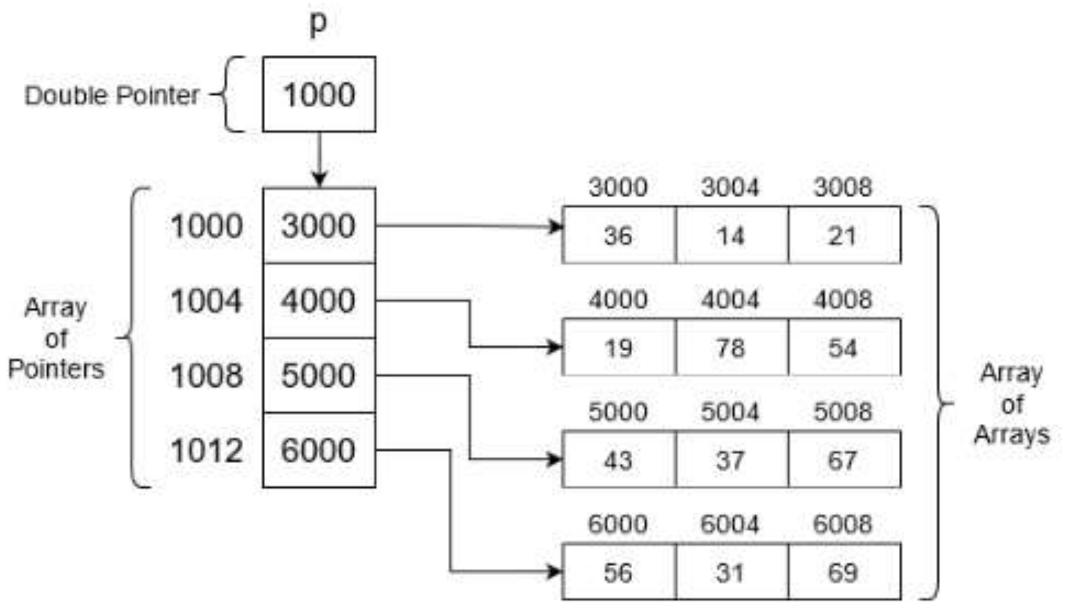
```
#include<stdio.h>

int  main()
{

    int i = 3;
    int *j;   //pointer to integer
    int **k;  // pointer to pointer (double indirection)

    // assign value to pointer and double pointer
    j = &i;
    k = &j;

    // ouput results

    printf(" the value of i is %d\n", i);
    printf("the value of j is %p\n", j);
    printf ("the value of k is %p\n\n", k);

    // output addresses

   printf(" the address if i is: %p \n", &i);
   printf(" the address of j is %p\n", &j);
   printf(" the address of k is %p\n", &k);

    // using indirection to get value of i;

    printf(" the value of i is %d\n", i);
    printf("the value of *j is %d\n", *j);
    printf("the value of k is %p\n", k);
    printf("the value of *k is %p\n", *k);
    printf ("the value of **k is %d\n", **k);

   return 0;
-- INSERT --                                                    6,5           Top
```



```
denis.manley@apollo:~/OS2/week2$ ./L2Q3_double_pointer
 the value of i is 3
the value of j is 0x7ffd6a0f4e4c
the value of k is 0x7ffd6a0f4e50


 the address if i is: 0x7ffd6a0f4e4c
 the address of j is 0x7ffd6a0f4e50
 the address of k is 0x7ffd6a0f4e58
 the value of i is 3
the value of *j is 3
the value of k is 0x7ffd6a0f4e50
the value of *k is 0x7ffd6a0f4e4c
the value of **k is 3
denis.manley@apollo:~/OS2/week2$
```

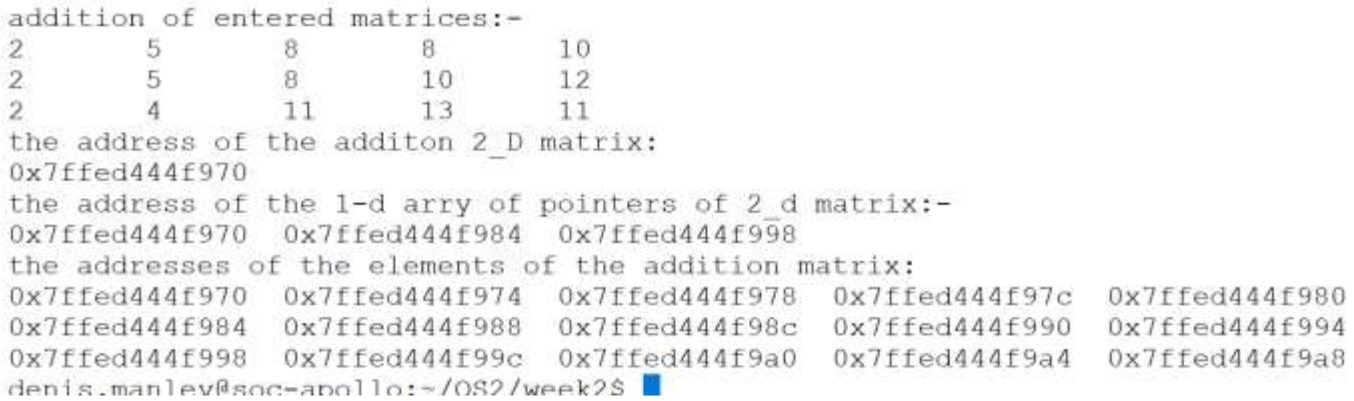
2 D Array

- A 2_D array is essentially :
- an arrays of pointers (variables that store an address)
- Each element of the 2_D array can be accessed using subscripts or pointers / double pointers



2D Matrix and Address

- The array name stores the address of the first element of 1_D array of pointers (double pointer (**) to single pointer(*)
- Each element of this 1-D array of “pointers” stores the address of the first element of an array of standard variables: a row of the 2_D a matrix.



```
addition of entered matrices:-
2       5       8       8       10
2       5       8       10      12
2       4       11      13      11
the address of the additon 2_D matrix:
0x7ffed444f970
the address of the 1-d arry of pointers of 2_d matrix:-
0x7ffed444f970  0x7ffed444f984  0x7ffed444f998
the addresses of the elements of the addition matrix:
0x7ffed444f970  0x7ffed444f974  0x7ffed444f978  0x7ffed444f97c  0x7ffed444f980
0x7ffed444f984  0x7ffed444f988  0x7ffed444f98c  0x7ffed444f990  0x7ffed444f994
0x7ffed444f998  0x7ffed444f99c  0x7ffed444f9a0  0x7ffed444f9a4  0x7ffed444f9a8
denis.manley@soc-apollo:~/OS2/week2$
```

Array of pointers

- 2_D array of integers:
  - **int matrix[3][4]**; //declare 2_D array
  - Access individual elements using *subscript notation*:
    - printf(“ value of row 2 col 2 is %d”, matrix[1][1]);

  - Actual Implementation: Access elements using pointer variables:
    - printf(“ value of row 2 col 2 is %d”, *(*(matrix + 1) + 1));
    - access the 1_D array of pointers *(matrix + 1)
    - Access an element of the of the 2_D array via *(*(matrix + 1) +1)

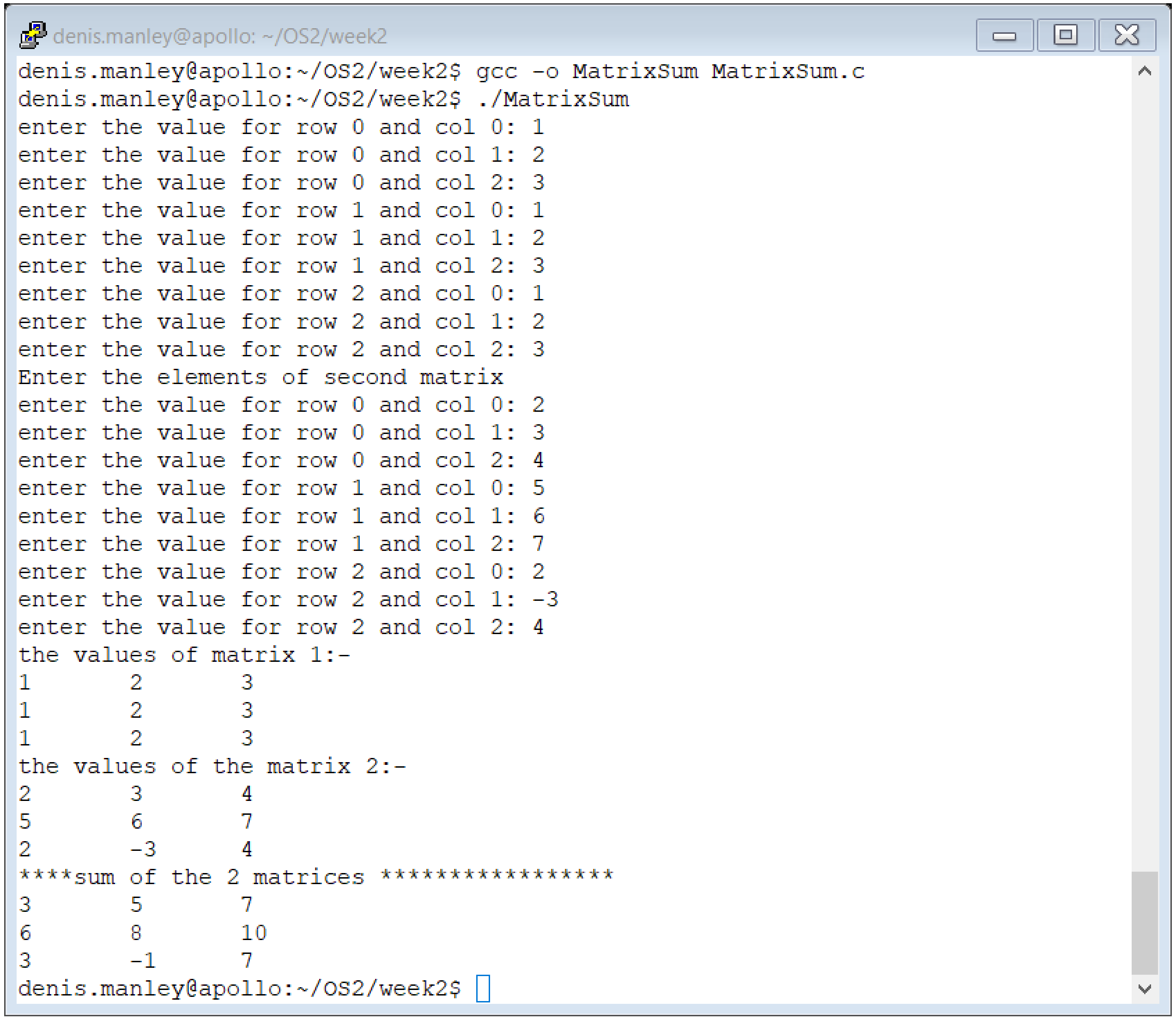
  - Explain *clearly* how the pointer notation prints row 2 col ?

Manipulate 2_D array

- To access each element you need to use nested for loops; consider a 3x3 matrix
- To print each value
  - for (row = 0; row<3; row++)
    - for (col = 0; col <3; col++)
      - printf(“%d “, matrix[row][col]); // can also use pointer arithmetic

- Class Question: How would you calculate and display the **addition** of two matrices.

Sample output Matrix Addition: (MatrixSum.c)

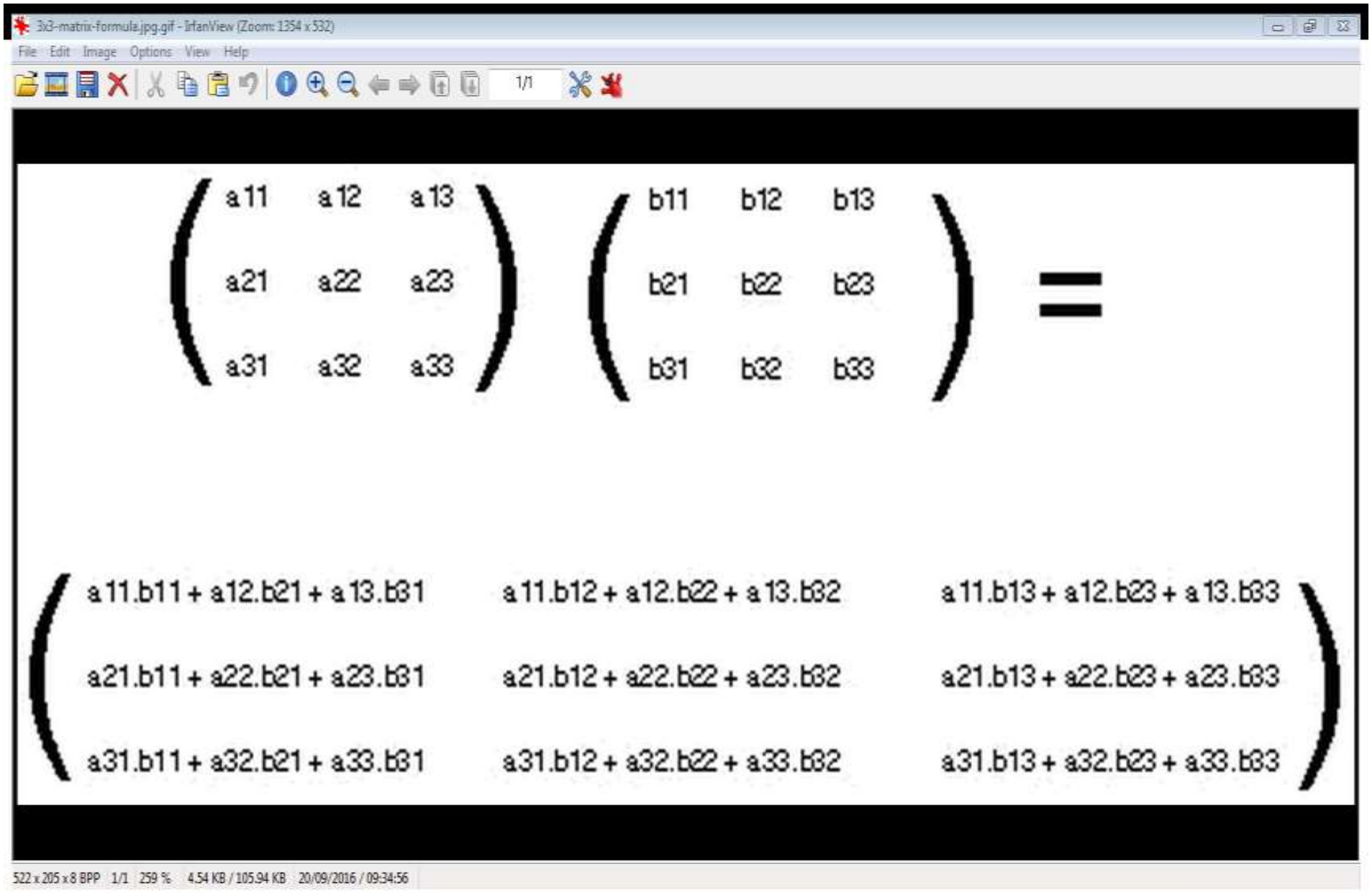


denis.manley@apollo: ~/OS2/week2

```
denis.manley@apollo:~/OS2/week2$ gcc -o MatrixSum MatrixSum.c
denis.manley@apollo:~/OS2/week2$ ./MatrixSum
enter the value for row 0 and col 0: 1
enter the value for row 0 and col 1: 2
enter the value for row 0 and col 2: 3
enter the value for row 1 and col 0: 1
enter the value for row 1 and col 1: 2
enter the value for row 1 and col 2: 3
enter the value for row 2 and col 0: 1
enter the value for row 2 and col 1: 2
enter the value for row 2 and col 2: 3
Enter the elements of second matrix
enter the value for row 0 and col 0: 2
enter the value for row 0 and col 1: 3
enter the value for row 0 and col 2: 4
enter the value for row 1 and col 0: 5
enter the value for row 1 and col 1: 6
enter the value for row 1 and col 2: 7
enter the value for row 2 and col 0: 2
enter the value for row 2 and col 1: -3
enter the value for row 2 and col 2: 4
the values of matrix 1:-
1       2       3
1       2       3
1       2       3
the values of the matrix 2:-
2       3       4
5       6       7
2       -3      4
****sum of the 2 matrices *****************
3       5       7
6       8       10
3       -1      7
denis.manley@apollo:~/OS2/week2$
```

Matrix multiplication

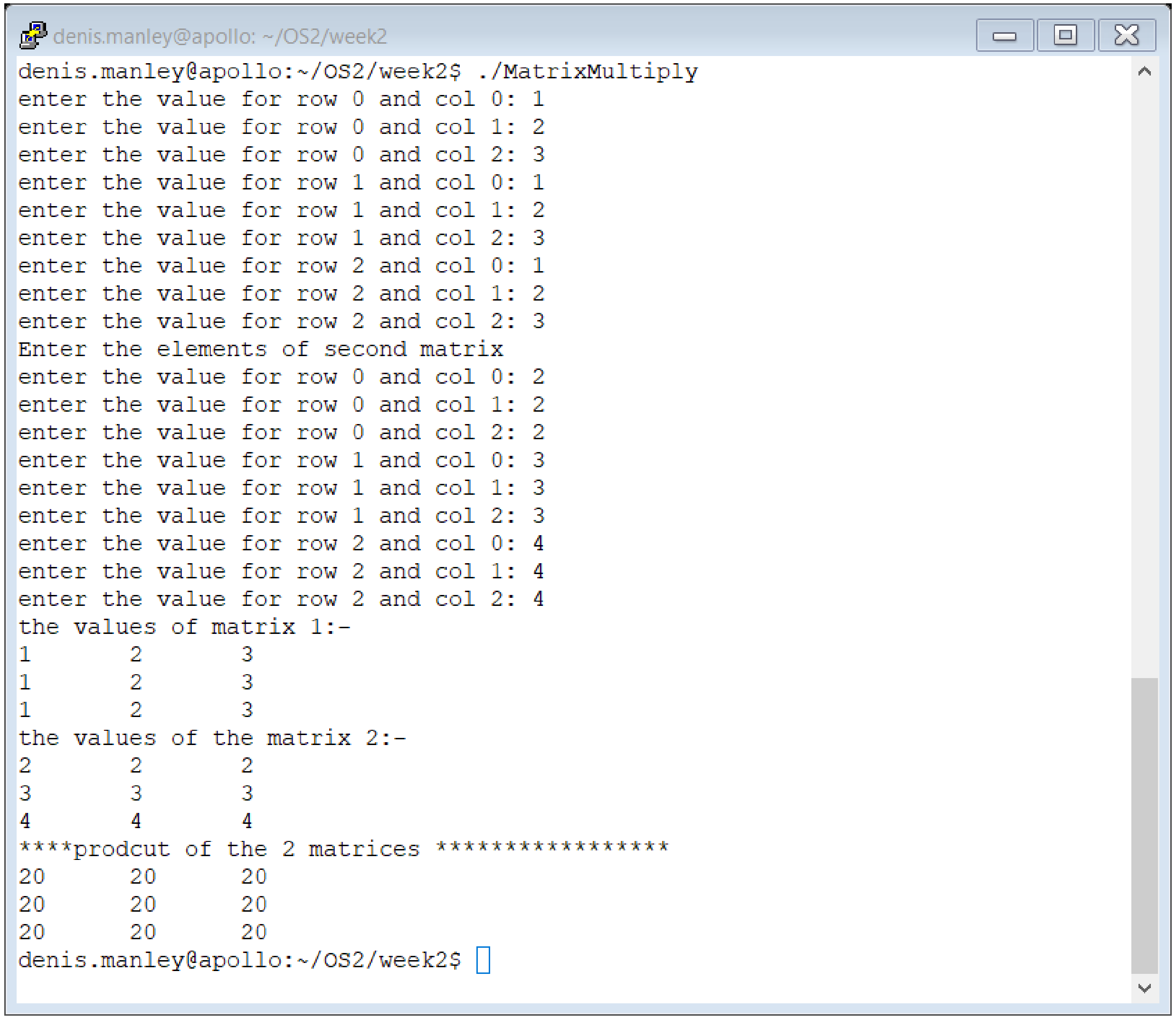
- How would you perform matrix multiplication of two matrices?
- The following link describes how to multiply two matrices: matrix multiplication
- Refer to the 3x3 image file on web courses


3x3-matrix-formula.jpg.gif - IrfanView (Zoom: 1354 x 532)
File Edit Image Options View Help
1/1
522 x 205 x 8 BPP 1/1 259 % 4.54 KB / 105.94 KB 20/09/2016 / 09:34:56

$$\begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{pmatrix} \begin{pmatrix} b11 & b12 & b13 \\ b21 & b22 & b23 \\ b31 & b32 & b33 \end{pmatrix} =$$

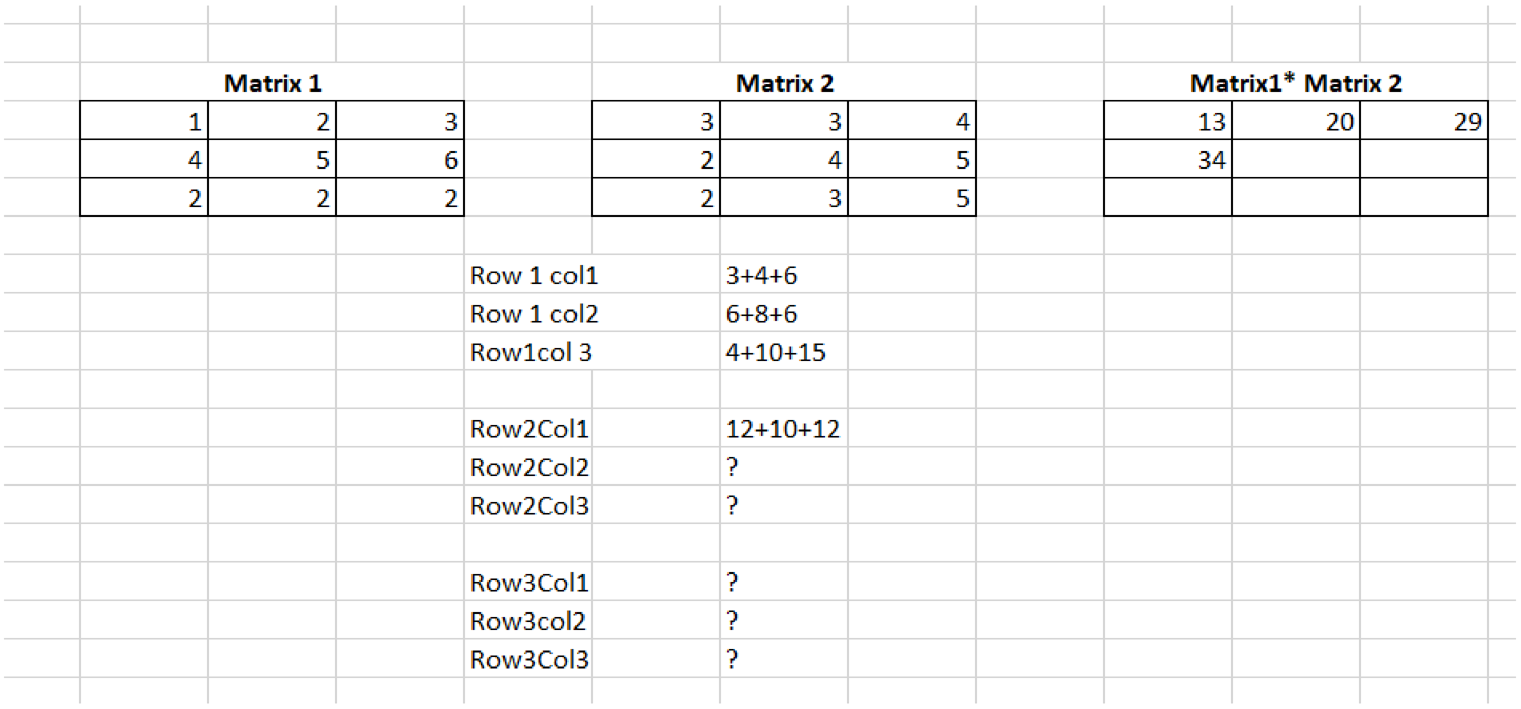
$$\begin{pmatrix} a11.b11 + a12.b21 + a13.b31 & a11.b12 + a12.b22 + a13.b32 & a11.b13 + a12.b23 + a13.b33 \\ a21.b11 + a22.b21 + a23.b31 & a21.b12 + a22.b22 + a23.b32 & a21.b13 + a22.b23 + a23.b33 \\ a31.b11 + a32.b21 + a33.b31 & a31.b12 + a32.b22 + a33.b32 & a31.b13 + a32.b23 + a33.b33 \end{pmatrix}$$

Sample output



```
denis.manley@apollo:~/OS2/week2$ ./MatrixMultiply
enter the value for row 0 and col 0: 1
enter the value for row 0 and col 1: 2
enter the value for row 0 and col 2: 3
enter the value for row 1 and col 0: 1
enter the value for row 1 and col 1: 2
enter the value for row 1 and col 2: 3
enter the value for row 2 and col 0: 1
enter the value for row 2 and col 1: 2
enter the value for row 2 and col 2: 3
Enter the elements of second matrix
enter the value for row 0 and col 0: 2
enter the value for row 0 and col 1: 2
enter the value for row 0 and col 2: 2
enter the value for row 1 and col 0: 3
enter the value for row 1 and col 1: 3
enter the value for row 1 and col 2: 3
enter the value for row 2 and col 0: 4
enter the value for row 2 and col 1: 4
enter the value for row 2 and col 2: 4
the values of matrix 1:-
1       2       3
1       2       3
1       2       3
the values of the matrix 2:-
2       2       2
3       3       3
4       4       4
****prodcut of the 2 matrices *****************
20      20      20
20      20      20
20      20      20
denis.manley@apollo:~/OS2/week2$
```

Matrix Multiplication exercise



Matrix 1

1	2	3
4	5	6
2	2	2

Matrix 2

3	3	4
2	4	5
2	3	5

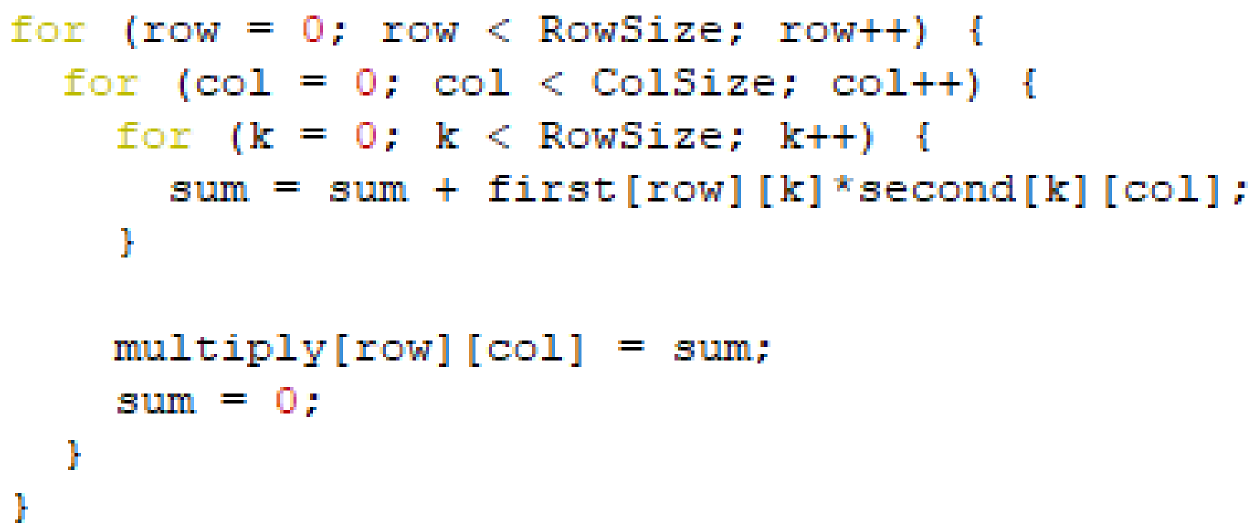
Matrix1* Matrix 2

13	20	29
34		

Row 1 col1	3+4+6
Row 1 col2	6+8+6
Row1col 3	4+10+15
Row2Col1	12+10+12
Row2Col2	?
Row2Col3	?
Row3Col1	?
Row3col2	?
Row3Col3	?

3 For loop: matrix multiplication

- Sample C code to multiplying 2_D matrices of the same dimensions :



```
for (row = 0; row < RowSize; row++) {
  for (col = 0; col < ColSize; col++) {
    for (k = 0; k < RowSize; k++) {
      sum = sum + first[row][k]*second[k][col];
    }

    multiply[row][col] = sum;
    sum = 0;
  }
}
```

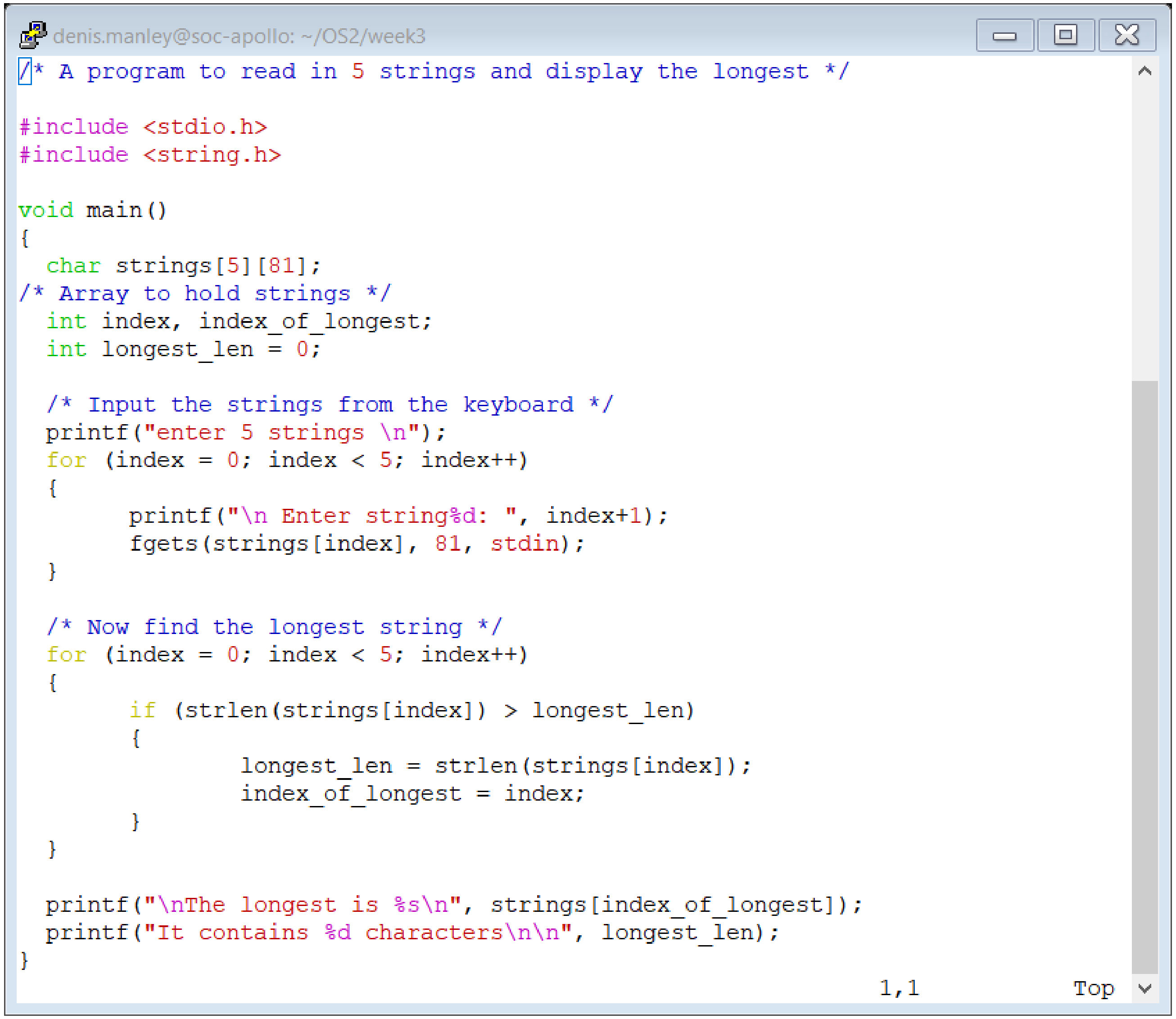
Array of string pointers

- If you wanted to create an array to store all the months [strings] of the year you could
  - Create a **2_d character array**: e.g. **char month[12][10]**
  - Why would this be inefficient? (hint: sparsity)

  - Another option is to create an **Array of Characters Pointers**: e.g. **char *months[12]**
  - Each element is then treated an a char * (essentially a string)
  - Remember you must assign memory for each using char *months[0] = “January”
  - The address of J is assigned to element 0
  - Consider the following program

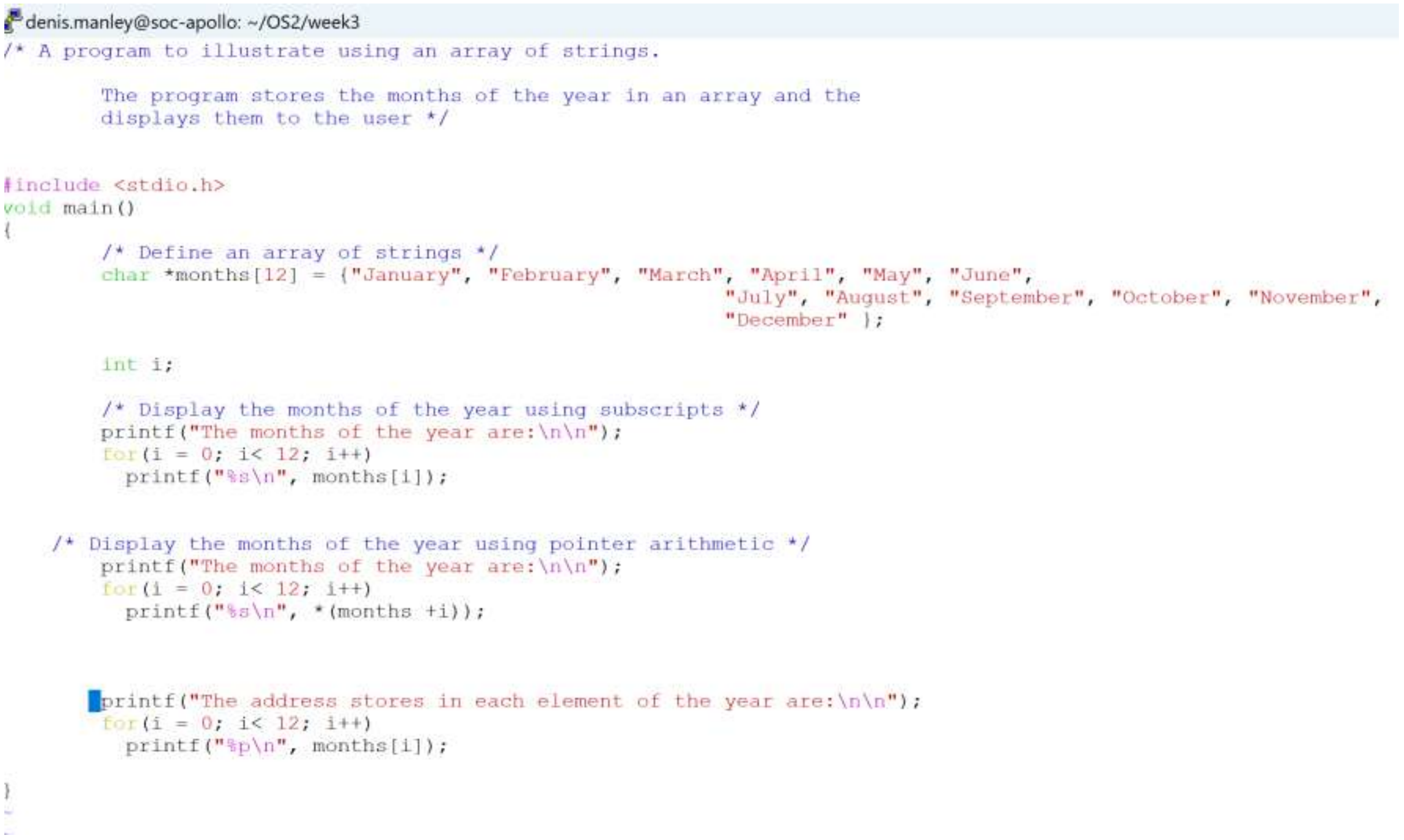
2-D character array (LONGSTR.C)

- Input strings (using 2-D character array)



```
/* A program to read in 5 strings and display the longest */

#include <stdio.h>
#include <string.h>

void main()
{
  char strings[5][81];
/* Array to hold strings */
  int index, index_of_longest;
  int longest_len = 0;

  /* Input the strings from the keyboard */
  printf("enter 5 strings \n");
  for (index = 0; index < 5; index++)
  {
        printf("\n Enter string%d: ", index+1);
        fgets(strings[index], 81, stdin);
  }

  /* Now find the longest string */
  for (index = 0; index < 5; index++)
  {
        if (strlen(strings[index]) > longest_len)
        {
                longest_len = strlen(strings[index]);
                index_of_longest = index;
        }
  }

  printf("\nThe longest is %s\n", strings[index_of_longest]);
  printf("It contains %d characters\n\n", longest_len);
}
                                                                  1,1           Top
```

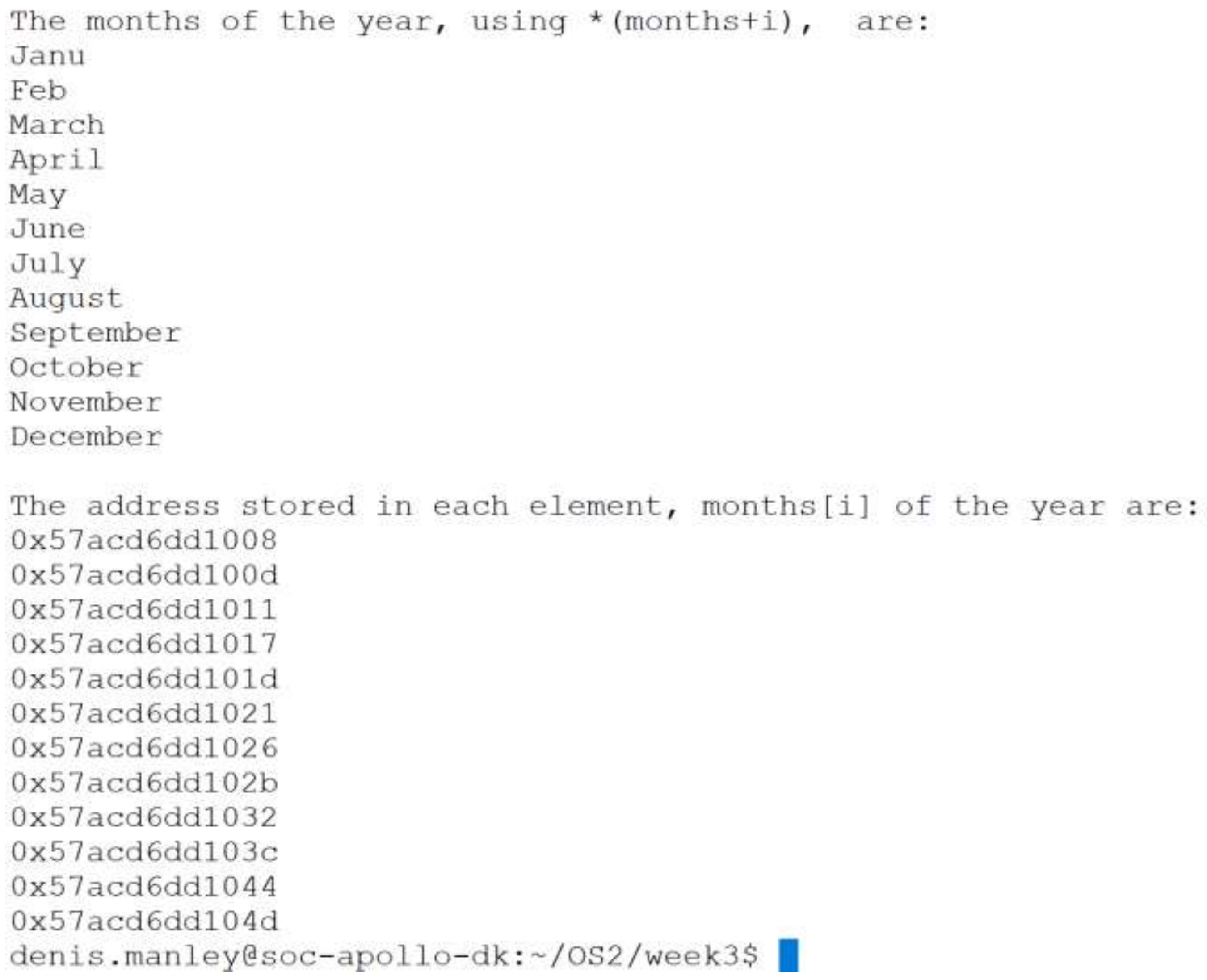
Array of string: **months.c**



denis.manley@soc-apollo: ~/OS2/week3

```c
/* A program to illustrate using an array of strings.

        The program stores the months of the year in an array and the
        displays them to the user */


#include <stdio.h>
void main()
{
        /* Define an array of strings */
        char *months[12] = {"January", "February", "March", "April", "May", "June",
                                                          "July", "August", "September", "October", "November",
                                                          "December" };

        int i;

        /* Display the months of the year using subscripts */
        printf("The months of the year are:\n\n");
        for(i = 0; i< 12; i++)
          printf("%s\n", months[i]);


    /* Display the months of the year using pointer arithmetic */
        printf("The months of the year are:\n\n");
        for(i = 0; i< 12; i++)
          printf("%s\n", *(months +i));



       printf("The address stores in each element of the year are:\n\n");
        for(i = 0; i< 12; i++)
          printf("%p\n", months[i]);

}
~
~
```

Partial Sample Output of months.c



```
The months of the year, using *(months+i),  are:
Janu
Feb
March
April
May
June
July
August
September
October
November
December

The address stored in each element, months[i] of the year are:
0x57acd6dd1008
0x57acd6dd100d
0x57acd6dd1011
0x57acd6dd1017
0x57acd6dd101d
0x57acd6dd1021
0x57acd6dd1026
0x57acd6dd102b
0x57acd6dd1032
0x57acd6dd103c
0x57acd6dd1044
0x57acd6dd104d
denis.manley@soc-apollo-dk:~/OS2/week3$
```

Home work

- Explain using the above code (longstr.c and months.c), and with a suitable example, why a character pointer array is more efficient than a 2_D array of characters in terms of the memory allocated. (Hint: show the memory allocated for the 2_D array and compare with the address shown in the previous slide.