GNG1106 – Fundamentals of Engineering Computation
Practice Questions

Short Answer Questions

1) Provide a short definition for a computer array.

2) What is the main difference between a C array and a C structure?

3) Circle any of the following terms that relate directly to the C function (includes function calls and function definitions) (2 points):

`Repeat while`	`variable`	`arguments`
`condition`	`#define`	`Logical expression`
`prototype`	`return`	`if-else`

4) Which type indicates that a function does not return any value to the calling function?

5) What is output by the following C program? (4 points)

```
void main()
{
 int i, j;

 for(i=0;i<5;i=i+1)
 {
       for(j=0;j<3;j=j+1)
           printf("%d", i+j);

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

Output

6) Given an integer variable `num`, provide the C instruction(s) that assigns a random integer value between 0 and 9 to the variable `num`.

7) Write the C declaration of variable *str* which is a pointer to a character string.

8) When the name of the structure variable is used as an argument in a function call, will its value or its reference be passed to the called function?

9) In C, which character is used to terminate a character string?

10) Circle the errors (syntax and logical) in the following pieces of code, and propose corrections in the adjacent box (only the lines changed need be shown – it is not necessary to rewrite the complete code):

i)

```
if(x==3)
      printf(”X is greater than 3”);
else
      printf(”X is less than 3”);
```

ii)

```
int main()
{
      int i;
      for (i=5,i>0,i=i-1);
            printf(“%d”, i);
      return(0);
}
```

iii)

```
#include <stdio.h>

int main ()
{
   int array (5) = {1, 2, 7, 4, 12};
   int n, result=0;
 /* sum the elements of the array */
   n = 1;
   while(n >= 5)
   {
        result = result + array(n);
        n = n + 1;
   }
   printf(“Sum is %f”, result);
   return (0);
}
```

11) For each of the following expressions, write the value produced when evaluated by the computer. Variables are initialized in the declarations below. The logical expression values give one of the symbolic constants TRUE (1) or FALSE (0) (5 points):

```
int w=1, x=2;
float y=1.0, z=2.0;
float a[5] = {1, 2.5, 3, 4.1, 5};
```

Example: `x+1` `___3___`

Expression	**Value**
`w % x`	_____
`a[2] >= z && y <= a[0]`	_____
`w + x/2`	_____
`(a[4]+5.1) >= 6.5 || a[1] != y`	_____
`w<=x != y<=z`	_____

12) Which of the following is a correct define directive?

```
define speed of light 30000
#define LONG_NUMBER 12345678901234567890
#define SHORT 0.01;
#DEFINE RADIUS 30
```

13) Define a new type *DATA_POINTS* that for a C structure with the following members:

- *identifier* (a string of up to MAX_STR_SIZE characters),
- *numPoints* (an integer that gives the number of valid data points),
- *points* (an array of the structure type POINT of size MAX_POINT_SIZE.

14) Is the following a structure type definition or a structure variable declaration?

```
#define MAXVALUES 17
struct valuesStr
{
   int num;
   double values[MAXVALUES];
};
```

15) Write a C function `swap()` that receives two addresses (to two integer values) in its parameters (two pointers) and swaps the values referenced to by the pointers. The function should have a type `void` (i.e. returns no value).

16) Provide the output of the following C code in the adjacent box?

```
main( )
{
   int pecans,apples;
   pecans = 100;
   apples = 101;
   printf("Values are %d %d\n",
           pecans, apples);
   fixup(pecans,&apples);
   printf("Now, they are %d %d\n",
           pecans, apples);
}

void fixup(int nuts, int *fruit)
{
   nuts = 135;
   *fruit = 172;
}
```

Output Screen

17) Given the type `VALUES` shown below and the declaration `VALUES *vPtr`, what expression provides access to the 5th element of the `values` member of the structure referenced by `vPtr` presented in question c.

```
#define MAXVALUES 17
typedef struct
{
   int num;
   double values[MAXVALUES];
} VALUES;
```

18) When the name of the structure variable is used as an argument in a function call, will its value or its reference be passed to the called function?

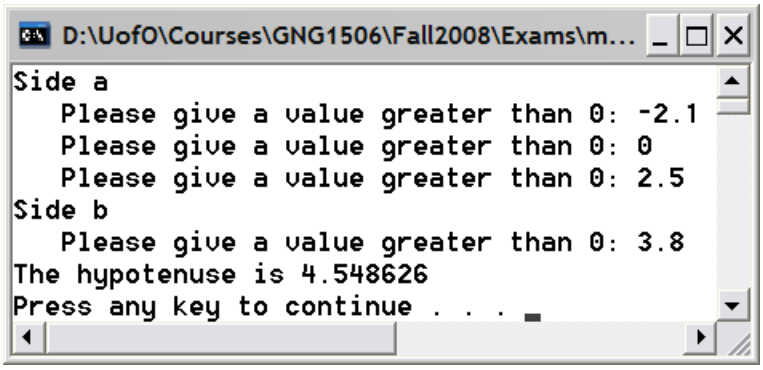
19) Consider the following code segment.

```
int *zPtr;
int *sPtr=NULL;
int integer, i;
int z[5] = {1, 2, 3, 4, 5};
sPtr = z;
for (i = 0; i <= 6; i++)
     printf("%d ", *(sPtr+i));
```

Find and describe the error in the loop structure.

Programming Model Questions

1) The following shows the output generated during an execution of the program shown on the next page:



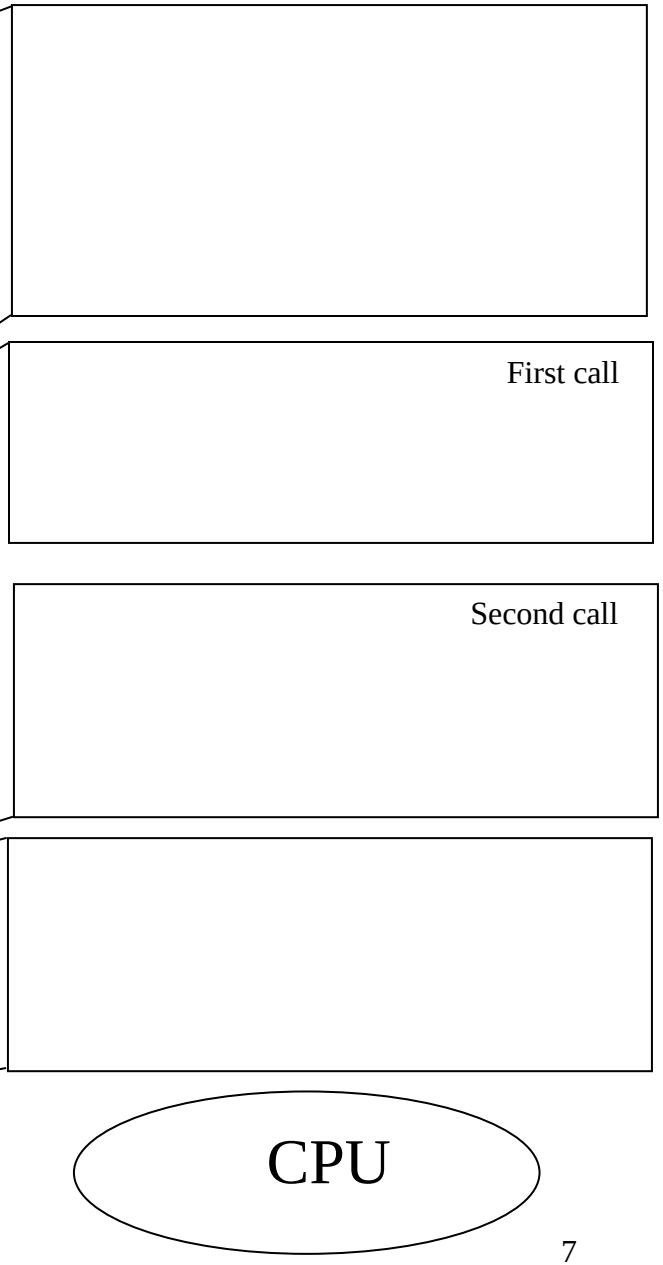
On the following page, show how the contents of the working memory are changed by the execution of the program. Record successive assignments to variables/parameters as follows:

Variabl	~~2~~, ~~6~~, ~~4~~, 10

Program Memory

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0
double getSide(void);
double computeHyp(double, double);
/*-----------------------------------------------*/
void main()
{
    double sideA, sideB; // Sides of triangle
    printf("Side a\n");
    sideA = getSide();
    printf("Side b\n");
    sideB = getSide();
    printf("The hypotenuse is %f\n",
      computeHyp(sideA, sideB));
}
/*-----------------------------------------------*/
double getSide(void)
{
    int flag;     // Becomes FALSE on valid value
    double side;  // Value for the side of a triangle
    do
    {
        printf("   Please give a value greater than 0: ");
        scanf("%lf",&side);
        if(side > 0) flag = FALSE;
        else flag = TRUE;
    } while(flag == TRUE);
    return(side);
}
/*-----------------------------------------------*/
double computeHyp(double x, double y)
{
   double hyp;
   hyp = x*x + y*y;
   hyp = sqrt(hyp);
   return hyp;
}
/*-----------------------------------------------*/
```

Working Memory



2) For the program on the following page:

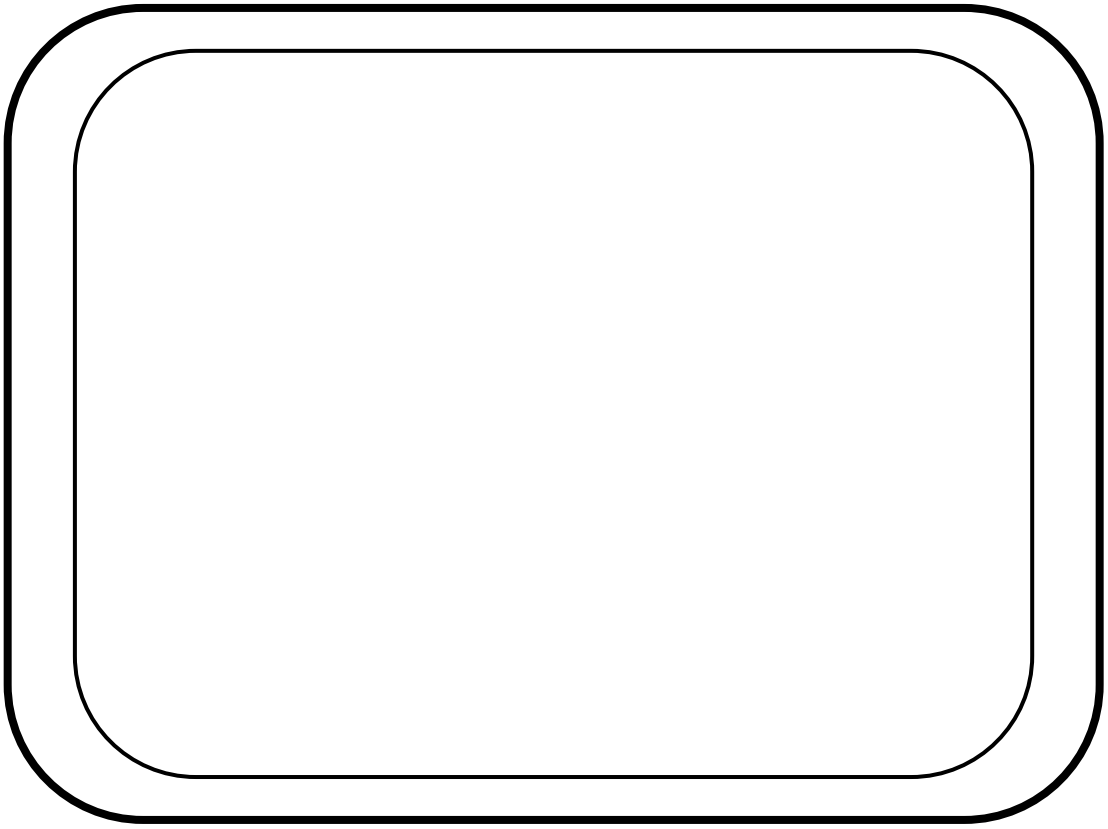
- The function `print1D` is called twice, and thus a two pieces of working memory are associated to the function, once for each call.
  ** you must show explicitly in the working memory which parameters and which variables are used by the function for each call **
- The successive calls to `rand()` gives the following sequence of values :
  41, 18467, 6334, 26500, 18169, 15724, 11478, 29358, 26962, 24464

(a) Show how the contents of the working memory are changed by the execution of the program. Note that the numbers in parentheses give addresses of memory locations. Record successive assignments to a variable/parameter and array elements as follows (? represents an unknown value):

Variable name	~~?~~ ~~2~~ ~~6~~ ~~4~~ 10

(b) For pointer variables, represent the values using adr1, adr2, adr3, etc. and draw a dashed line to the memory location referenced by the address value.

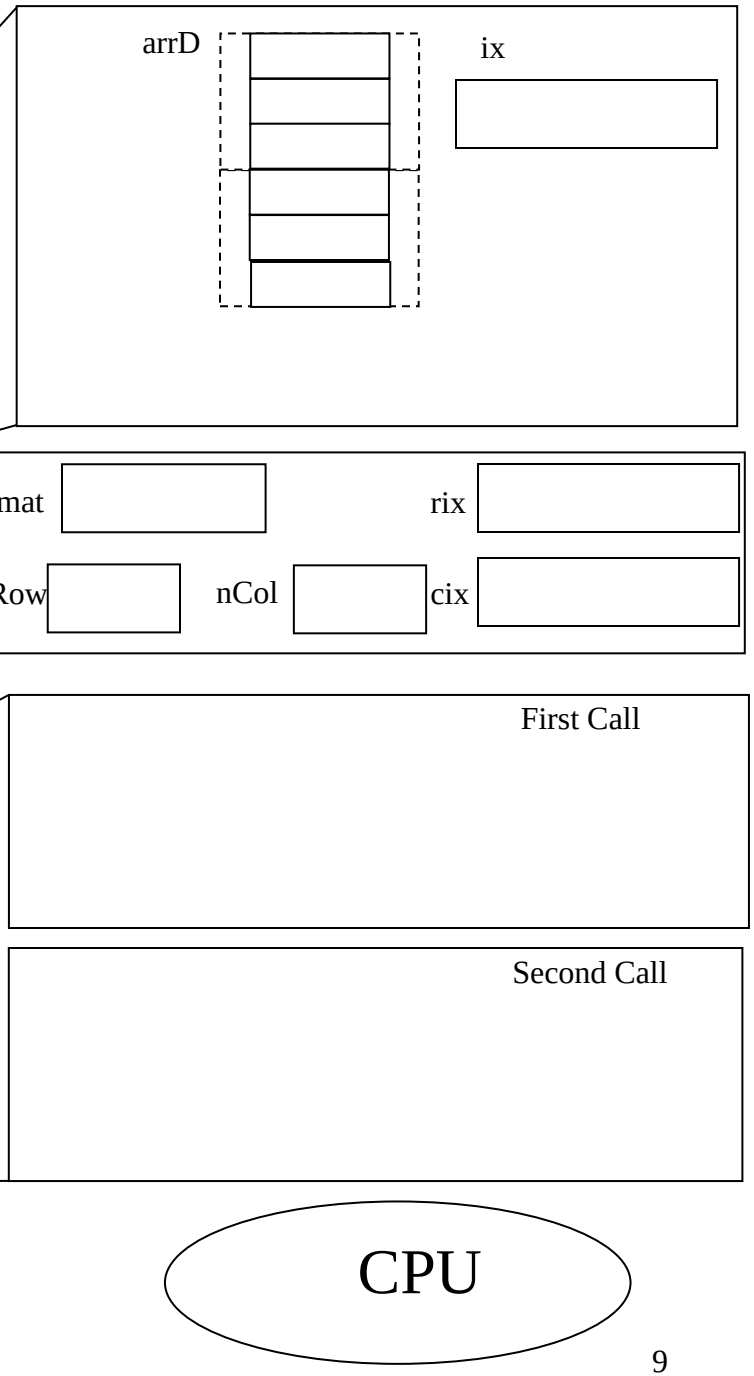
(c) Record below what the terminal screen shows when the program is executed, including your input.



Program Memory

```
#include <stdio.h>
#include <stdlib.h>
#define NROWS 2
#define NCOLS 3
void fill_random(int, int ncols, int[][ncols]);
void print1D(int, int[]);
/*-----------------------------------------------*/
void main()
{
    int arr2D[NROWS][NCOLS];
    int ix;

    fill_random (NROWS, NCOLS, arr2D);
    for(ix=0 ; ix < NROWS ; ix = ix + 1)
        print1D(NCOLS, arr2D[ix]);
}
/*-----------------------------------------------*/
void fill_random(int nRow, int nCol,
                  int mat[][nCol])
{
   int rix, cix;  /* index for rows and columns */
   for(rix = 0 ; rix <= nRow - 1; rix = rix + 1)
      for(cix = 0 ; cix <= nCol-1 ; cix = cix + 1)
          mat[rix][cix] = 5 + rand()%10 ;
}
/*-----------------------------------------------*/
void print1D(int num, int arr[])
{
   int ix;  /* index for array */
   for(ix = 0 ; ix < num; ix = ix + 1)
   {
     printf("%3d", arr[ix]);
   }
   printf("\n");
}
/*-----------------------------------------------*/
```

Working Memory



3) For the program on the following page:

(a) Show how the contents of the working memory are changed by the execution of the program. Record successive assignments to a variable/parameter as follows (? represents an unknown value):

Variable	~~?~~ ~~2~~ ~~6~~ ~~4~~ 10

(b) In the case of character arrays, show all characters in the arrays (even those after the nul character) and the changes made to the contents of the array as follows:

Array	‘a’ ‘b’ ‘c’ ‘d’ ~~‘e’~~ ‘\0’ ‘f’ ‘g’ ‘\0’ ‘\0 ‘\0’

(c) For pointer variables, represent the values using adr1, adr2, adr3, etc. and draw a dashed line to the memory location referenced by the address value.

(d) On the terminal screen, show the output of the program, i.e., what is printed on the screen.

Program Memory

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define LEN 10
void print1(char *, int );
void print2(char [][], int, int);

/*-----------------------------------------------*/
int main( )
{
   char days[7][LEN] = {"Monday","Tuesday","Wednesday",
            "Thursday","Friday","Saturday","Sunday"};

   print1(days[1], 4);
   print2(days, 4, 3);
   system("pause");
   return (0);
}

/*-----------------------------------------------*/
void print1(char *ptr, int size)
{
   char day[LEN]= "";

   strcpy(day, ptr);
   *(day+size) = '\0';
   puts(day);
}

/*-----------------------------------------------*/
void print2(char ptr[][LEN], int index, int size)
{
   char day[LEN]= "";

   strcpy(day, ptr[index]);
   *(day+size) = '\0';
   puts(day);
}
/*-----------------------------------------------*/
```

Terminal/Output

Working Memory

days

'M' 'o' 'n' 'd' 'a' 'y' '\0' '\0' '\0' '\0'
'T' 'u' 'e' 's' 'd' 'a' 'y' '\0' '\0' '\0'
'W' 'e' 'd' 'n' 'e' 's' 'd' 'a' 'y' '\0'
'T' 'h' 'u' 'r' 's' 'd' 'a' 'y' '\0' '\0'
'F' 'r' 'i' 'd' 'a' 'y' '\0' '\0' '\0' '\0'
'S' 'a' 't' 'u' 'r' 'd' 'a' 'y' '\0' '\0'
'S' 'u' 'n' 'd' 'a' 'y' '\0' '\0' '\0' '\0'

ptr

size

day

ptr

index

size

day

CPU

4) For the program on the following page:

Show how the contents of the working memory are changed by the execution of the program. Record successive assignments to a variable/parameter as follows (? represents an unknown value):

VariableName	~~?~~ ~~2~~ ~~6~~ ~~4~~ 10

In the case of character arrays, show all characters in the arrays (even those after the nul character) and the changes made to the contents of the array as follows:

Array name	‘a’ ‘b’ ‘c’ ‘d’ ~~‘e’~~ ‘f’ ‘g’ ‘\0’ ‘\0 ‘\0’ ‘\0’

For pointer variables, represent the values using adr1, adr2, adr3, etc. and draw a dashed line to the memory location referenced by the address value.

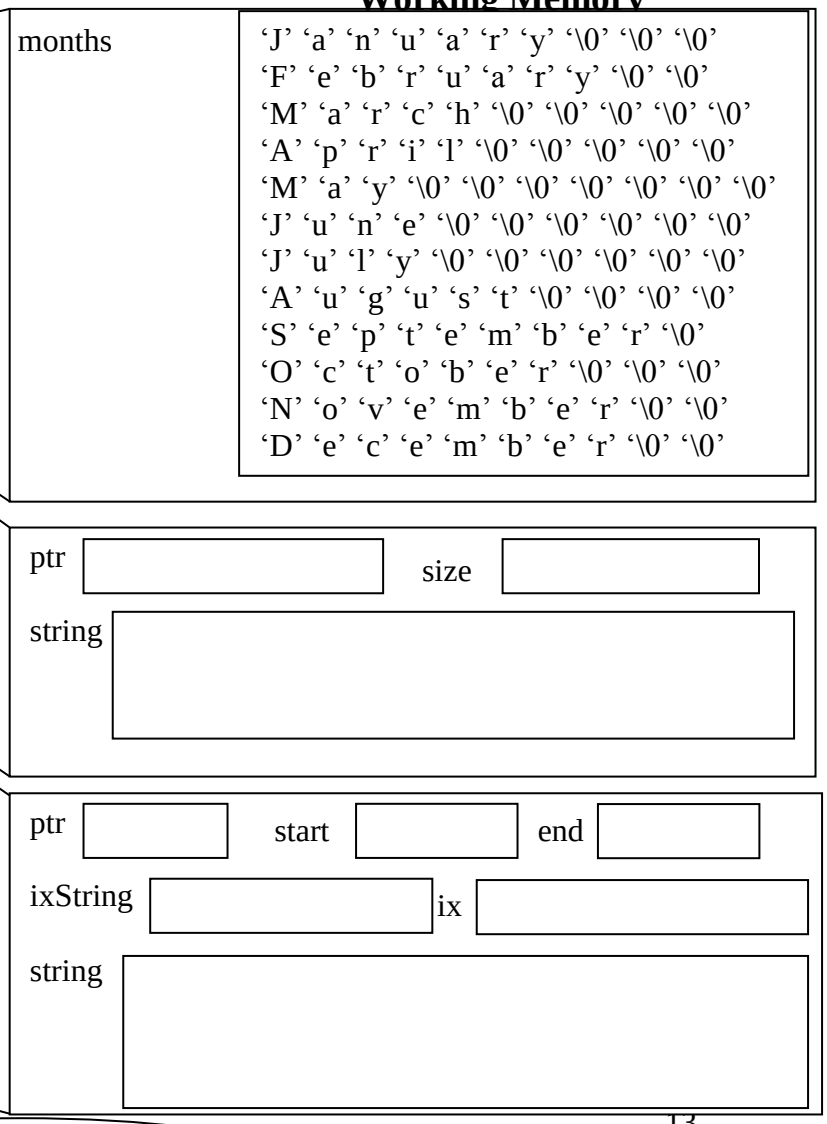
On the terminal screen, show the output of the program, i.e., what is printed on the screen.

Program Memory

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define LEN 10
void displayTruncated(char *, int );
void displaySubstring(char [][], int, int);

/*-----------------------------------------------*/
int main( )
{
   char months[12][LEN] = {"January","February","March",
            "April","May","June","July","August",
            "September","October","November","December"};
   displayTruncated(months[3], 4);
   displaySubstring(months[8], 2, 7);
   return (0);
}
/*-----------------------------------------------*/
void displayTruncated(char *ptr, int size)
{
   char string[LEN]= "";

   strcpy(string, ptr);
   string[size] = '\0';
   puts(string);
}
/*-----------------------------------------------*/
void displaySubstring(char *ptr, int start, int end)
{
   char string[LEN]= "";
   int ixString = 0, ix = start;
   while(ix <= end)
   {
       string[ixString] = ptr[ix];
       ixString = ixString + 1;
       ix = ix + 1;
   }
   string[ixString] = '\0';
   puts(string);
}
/*-----------------------------------------------*/
```

Terminal/Output

Working Memory



CPU

5) For the program on the following page:

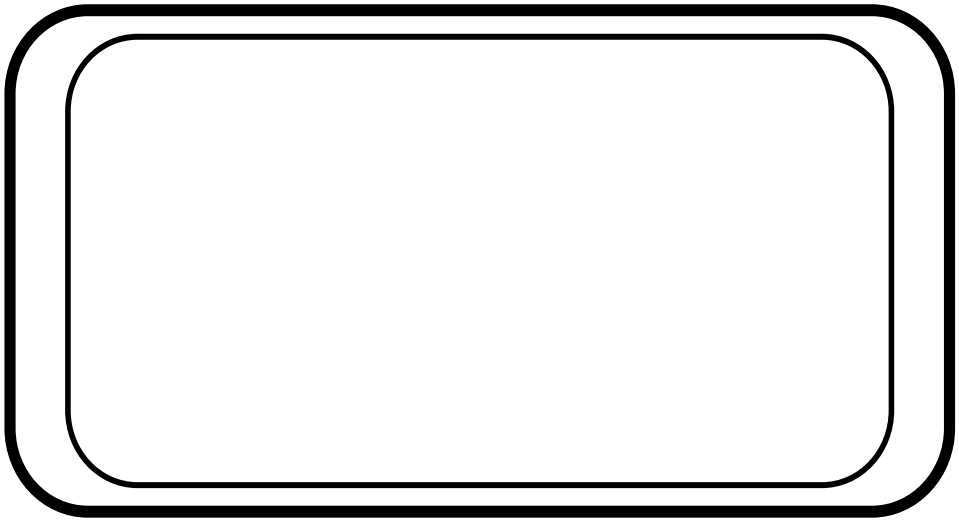
(d) Show how the contents of the working memory are changed by the execution of the program. Note that the numbers in parentheses give addresses of memory locations. Record successive assignments to a variable/parameter and array elements as follows (? represents an unknown value):

Variable name	~~?~~ ~~2~~ ~~6~~ ~~4~~ 10

(e) For pointer variables, represent the values using adr1, adr2, adr3, etc. and draw a dashed line to the memory location referenced by the address value.

(f) Record below what the terminal screen shows when the program is executed, including your input.

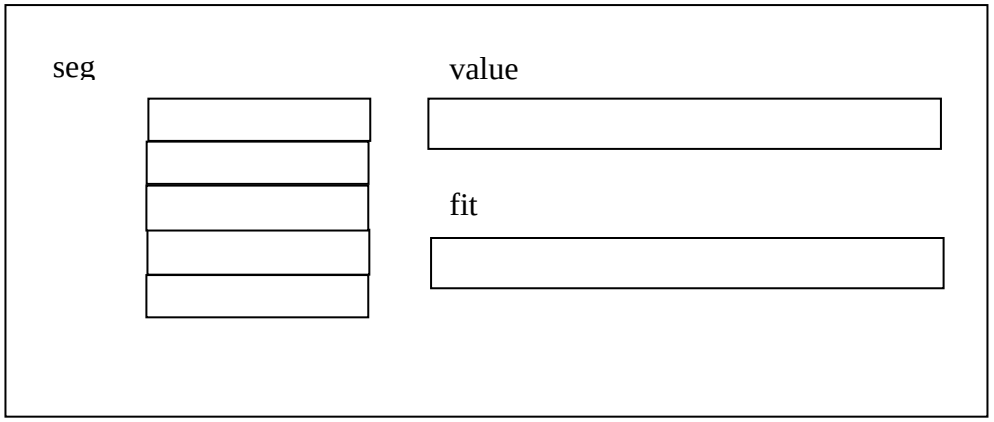
.



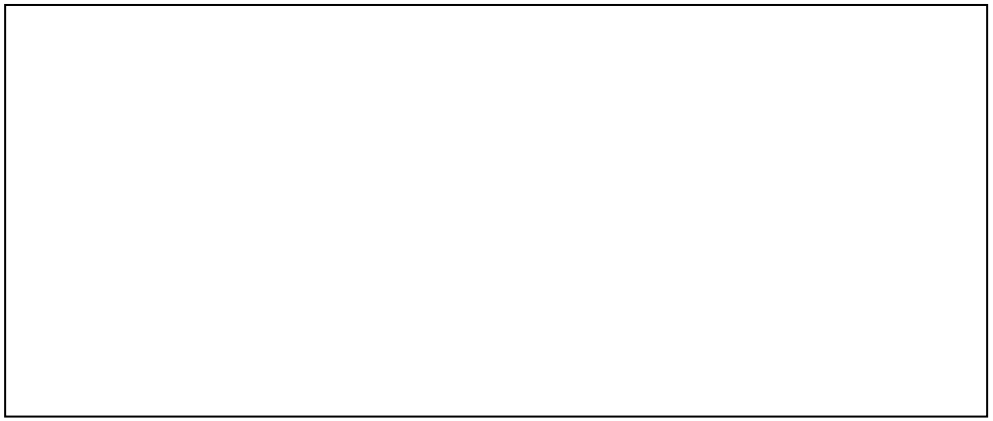
Program Memory

```
#include <stdio.h>
#define SIZE 5

double suprog(double *, double *, int);
double compress(double *, int);


void main()
{
   double value, fit, seg[SIZE] =
                          {3.0, 6.0, 4.0, 8.0, 5.0};
   printf("Enter value:\n");   /*Enter 0.5*/
   scanf("%lf",&value);
   fit=suprog(seg, &value, SIZE);
   printf("Array of SIZE %d gives %f\n", SIZE, fit);
}

double suprog(double *array, double *num, int size)
{
   int i;

   for (i = 1; i <= size - 1; i++)
        if (array[i-1] > array[i]){
            array[i-1] = (*num)*array[i];}
   printf("Array[%d]=%f\n",4, array[4]);

   return(compress(array, size));
}

double compress(double *arr, double size)
{
   int i;
   double temp;
   temp=1;
   for(i=0;i<size;i++){
        temp=temp*arr[i];}
   return(temp);
}
```

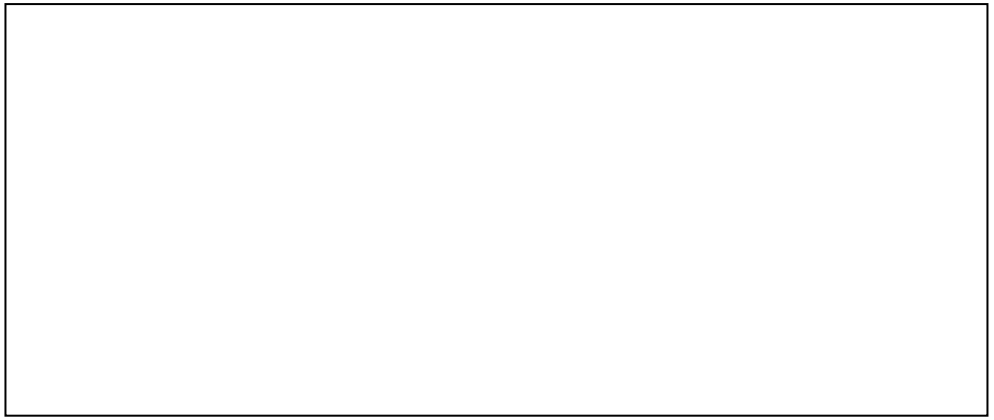
Working Memory (main)



Working Memory (suprog)



Working Memory (compress)



Programming Questions

1) In a global warming study, temperatures at the North Pole are recorded at noon every day. Each month the registered data is processed by software for subsequent analysis and research.

   (a) Complete the following function that computes the average monthly temperature.

```
/*-------------------------------------------------------------------------
Function: averageTemp
Parameters:
    numDays - number of daily temperatures stored in the array.
    temps - reference to an array that contains the temperatures
Returns:  Average temperature of the month.
Description: Computes the average number of temperatures stored in the
             referenced array.
-------------------------------------------------------------------------*/
double averageTemp(int numDays, double temps[])
{


}
```

   (b) Complete the following function that counts the number of warm days in a month.

```
/*-------------------------------------------------------------------------
Function: numHotDays
Parameters:
    thresholdTemp - threshold temprature for determining hot days
    numDays - number of daily temperatures stored in the array.
    temps - reference to an array that contains the temperatures
Returns:  Number of days where the temperature exceeded thresholdTemp.
Description: Scans the referenced arrays counting the number of
             temperatures whose value exceeded the threshold temperature.
-------------------------------------------------------------------------*/
int numHotDays(double thresholdTemp, int numDays, double temps[])
{


}
```

2) **Newton's Method for Calculating a Square Root**

This method involves a bit of multiplying and dividing but it will arrive at the precise square root over time and trials; it can be used to develop a program to compute the square root, sqr, of a number x.

- Initialise sqr = x.
- Update the value of the square root repeatedly using $sqr_{new} = (x + sqr_{old}^2)/(2sqr_{old})$
- Stop the updating sqr when the difference between sqr_{new} and sqr_{old} is less than x/10000.

Complete the following function that computes the square root of a given number using the above method. Consider coding the function in CodeBlocks and comparing the result with the standard `sqrt` math function.

```
/*-------------------------------------------------------------------------
Function: squareRoot
Parameters:
    x - value of x
Return:  The square root of x.
Description:  Computes the square root of x using the Newton Method.
-------------------------------------------------------------------------*/
double squareRoot(double x)
{

}
```

3) Computing the value of e

The value of e can be approximated using the following series:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + .. + \frac{1}{i!} + ... \frac{1}{n!} = 1 + \sum_{i=1}^{n} \frac{1}{n!}$$

Complete the following function that receives a value for *n* (i.e. defines the number of terms used to compute *e*) and returns a value for *e*. Note that the term 1/*i*! is simply 1/*i*(*i*-1)!, that is the i^{th} term can be computed from the $(i-1)^{th}$ term simply by dividing it by *i*. Use this fact in your solution.

```
/*-------------------------------------------------------------------------
Function: computeE
Parameters:
    n - gives number of terms to use to compute e
Return:  The value of e.
Description:  Computes the value of e using the n terms (added to 1).
-------------------------------------------------------------------------*/
double computeE(int n)
{

}
```

4) The Arrowhead Matrix

An "Arrowhead" matrix is a square matrix that contains zeros in all elements except for the first row, first column and the main diagonal. The non-zero elements must NOT contain the value zero. The following illustrates an arrowhead matrix (* represents a non-zero value):

$$\mathbf{A} = \begin{bmatrix} * & * & * & * \\ * & * & 0 & 0 \\ * & 0 & * & 0 \\ * & 0 & 0 & * \end{bmatrix}$$

Complete the following function that will receive a square matrix of integers and returns TRUE if the matrix represents an arrowhead matrix and FALSE otherwise. Your function must be as efficient as possible.

```
#define TRUE 1
#define FALSE 0
/*-------------------------------------------------------------------
Function: isArrowhead
Parameters:
    dim - dimension of square matrix
    mat - reference to a 2D array (matrix
Return:  TRUE if matrix is an arrow head matrix and FALSE otherwise.
Description:  Scans the elements of the matrix to ensure that first column
              firt row and the diagonal contain no zeros and all other
              elements contain zeros.
-------------------------------------------------------------------*/
int isArrowhead(int dim, int mat[][dim])
{

}
```

5) The Unit Matrix

An N X N square matrix is "**Unit Matrix**" when, all elements on the main diagonal, that is all the elements a_{ii} for $0 \leq i < N$ contain the value 1 AND all other elements in the matrix contain the value 0. The following 3 X 3 matrix is an example of a unit matrix.

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Complete the following function that will take any square matrix of integers and returns TRUE if the matrix represents a unit matrix and FALSE otherwise. Your function must be efficient.

```
#define TRUE 1
#define FALSE 0
/*-------------------------------------------------------------------
Function: isUnit
Parameters:
    dim - dimension of square matrix
    mat - reference to a 2D array (matrix
Return:  TRUE if matrix is a unit matrix and FALSE otherwise.
Description:  Scans the elements of the matrix to ensure that the diagonal
              elements contain 1 and all other elements contain 0.
-------------------------------------------------------------------*/
int isUnit(int dim, int mat[][dim])
{
```

```
}
```

(6) Transposing a matrix.

The transpose of the matrix (A^T) is such that element a_{rc} in the original matrix (A) will be element $a^T{}_{cr}$ in the transposed matrix. The number of rows in $\boldsymbol{A}$ becomes the number of columns in $\boldsymbol{A}^{\mathbf{T}}$, and the number of columns in A becomes the number of rows in $\boldsymbol{A}^{\mathbf{T}}$.

For example:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

To facilitate transposing of matrices, a structure is used to encapsulate a matrix as follows:

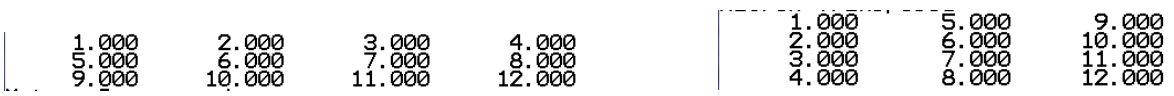
```
#define MAX_ROWS_COLS 100  // Allows matrices up to 100 rows and columns
typedef struct
{
      int nrows;  // number of rows used
      int ncols;  // number of columns used
      double mat[MAX_ROWS_COLS][MSX_ROWS_COLS];
} MATRIX;
```

Thus MATRIX type specifies a matrix of maximum size (100 X 100). The members `nrows` and `ncols` define the actual number of rows and number of columns used in the matrix. Such a type allows the transposition of a matrix provided that the dimensions do not exceed 100.

(a) Complete the following function that transposes a matrix. Note that the reference to the first matrix will be a well-defined matrix (that is, the values of the members are valid), while the second one (for storing the transpose) is not.

```
/*-------------------------------------------------------------------------
Function: transpose
Parameters:
    mat - reference to matrix to transposed
    matT - reference to matrix for storing the transpose
Description:  Transposes the matrix referenced by mat and stores
              the transpose into the matrix referenced by matT.
-------------------------------------------------------------------------*/
void transpose(MATRIX *mat, MATRIX *matT)
{

}
```

(b) Complete the following function that prints the matrix on the console. The output is formatted to show each row on one line, values aligned and 3 significant digits as shown (the examples show a 3 x 4 matrix printed and a 4 x 3 matrix printed.



```
 1.000   2.000   3.000   4.000
 5.000   6.000   7.000   8.000
 9.000  10.000  11.000  12.000
```

```
 1.000   5.000   9.000
 2.000   6.000  10.000
 3.000   7.000  11.000
 4.000   8.000  12.000
```

```
/*-------------------------------------------------------------------------
Function: printMat
Parameters:
    mat - reference to matrix
Description:  Displays formatted output of the matrix. Uses 3 significant
              digits in the output and aligns numbers.
-------------------------------------------------------------------------*/
void printMat(MATRIX *mat)
{

}
```