GNG1106 – Fundamentals of Engineering Computation
Practice Questions - Solutions

Short Answer Questions

1) A computer array is a collection of computer variables of the same type.
2) A C array is a collection of variables all of the same type while a C structure is a collection of variables of different types.
3)

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

4) void

5)

```
012
123
234
345
456
```

6) `num = rand()%10;`

7) `char *str;`

8) Its value.

9) The nul character, '\0'. Its decimal value is 0.

10)

i)

```
if(x==3)
      printf("X is greater than 3\n");
else
      printf("X is less than 3\n");
```

```
if(x > 3)
```

ii)

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

```
for (i=5;i>0;i=i-1)
```

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);
}
```

```
n = 0
while(n < 5)

result = result + array[n];


printf("Sum is %d", result);
```

11)

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

12) `#define LONG_NUMBER 12345678901234567890`

13)

```
typedef struct
{
   char identifier[MAX_STR_SIZE+1]; // +1 to accommodate '\0'
   int numPoints;
   POINT points[MAX_POINT_SIZE];
} DATA_POINTS;
```

14) Structure type definition

15)

```
void swap(int *n1, int *n2)
{
   int temp;
   temp = *n2;
   *n2 = *n1;
   *n1 = temp;
}
```

16)

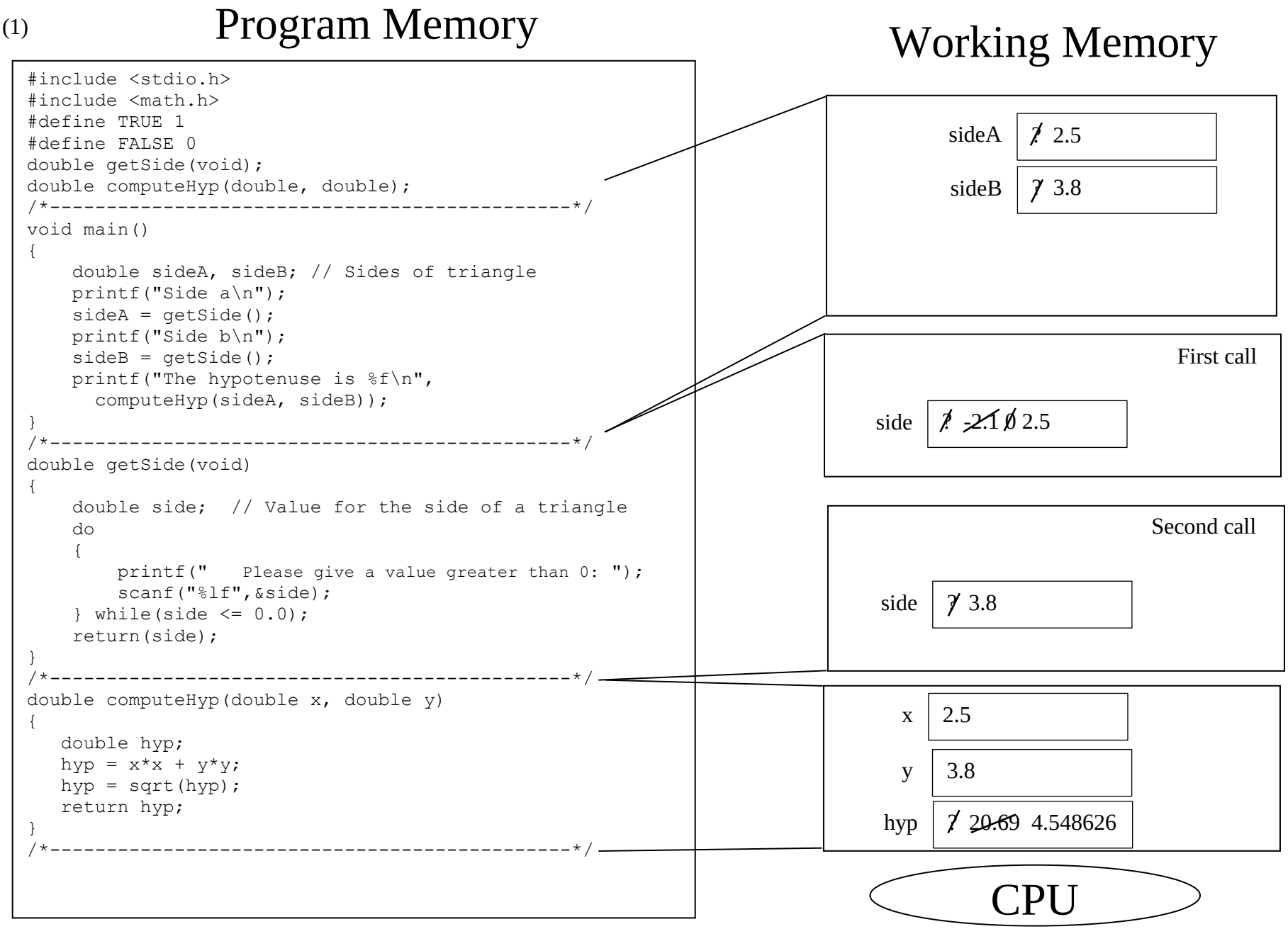
```
Values are 100 101
Now, they are 100 172
```

17) `vptr->values[4]`

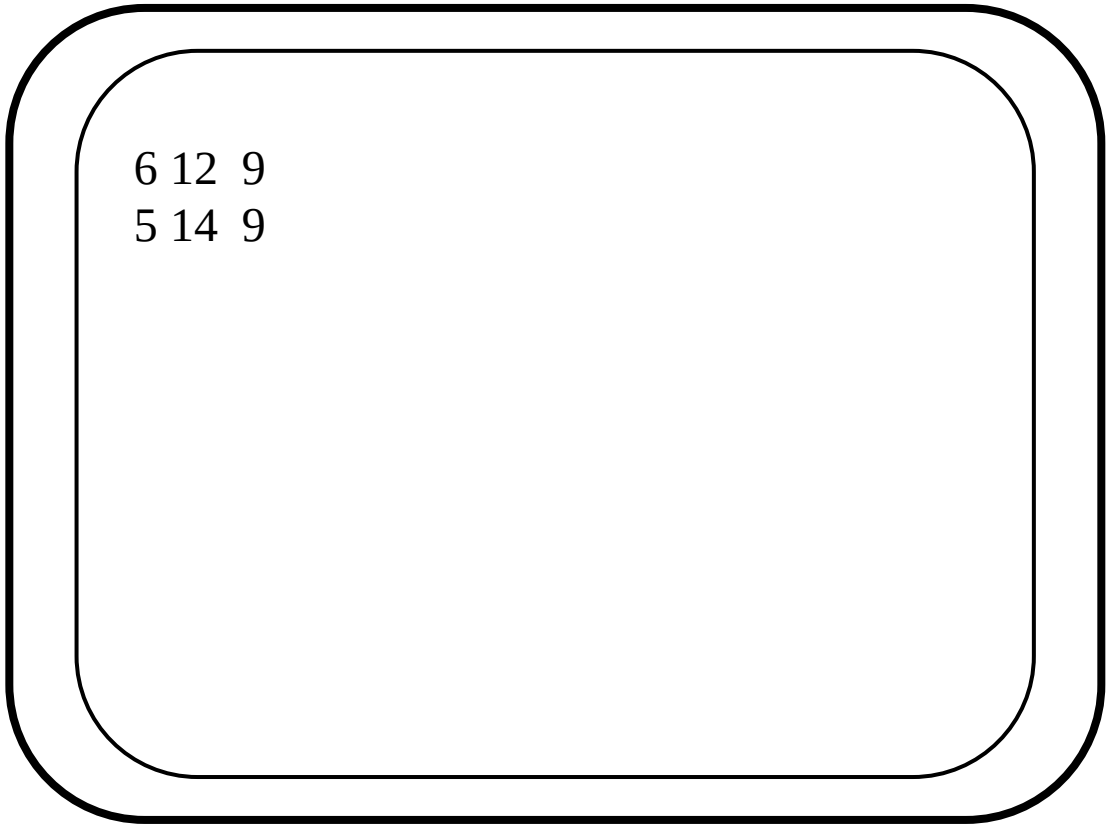
18) There are only 5 elements in the array z. The index is incremented past the end of this array, that is, 7 elements. The index i should only have been incremented to 4 instead of 6 (`i <= 4`) in the for loop.

Programming Model Questions

(1)



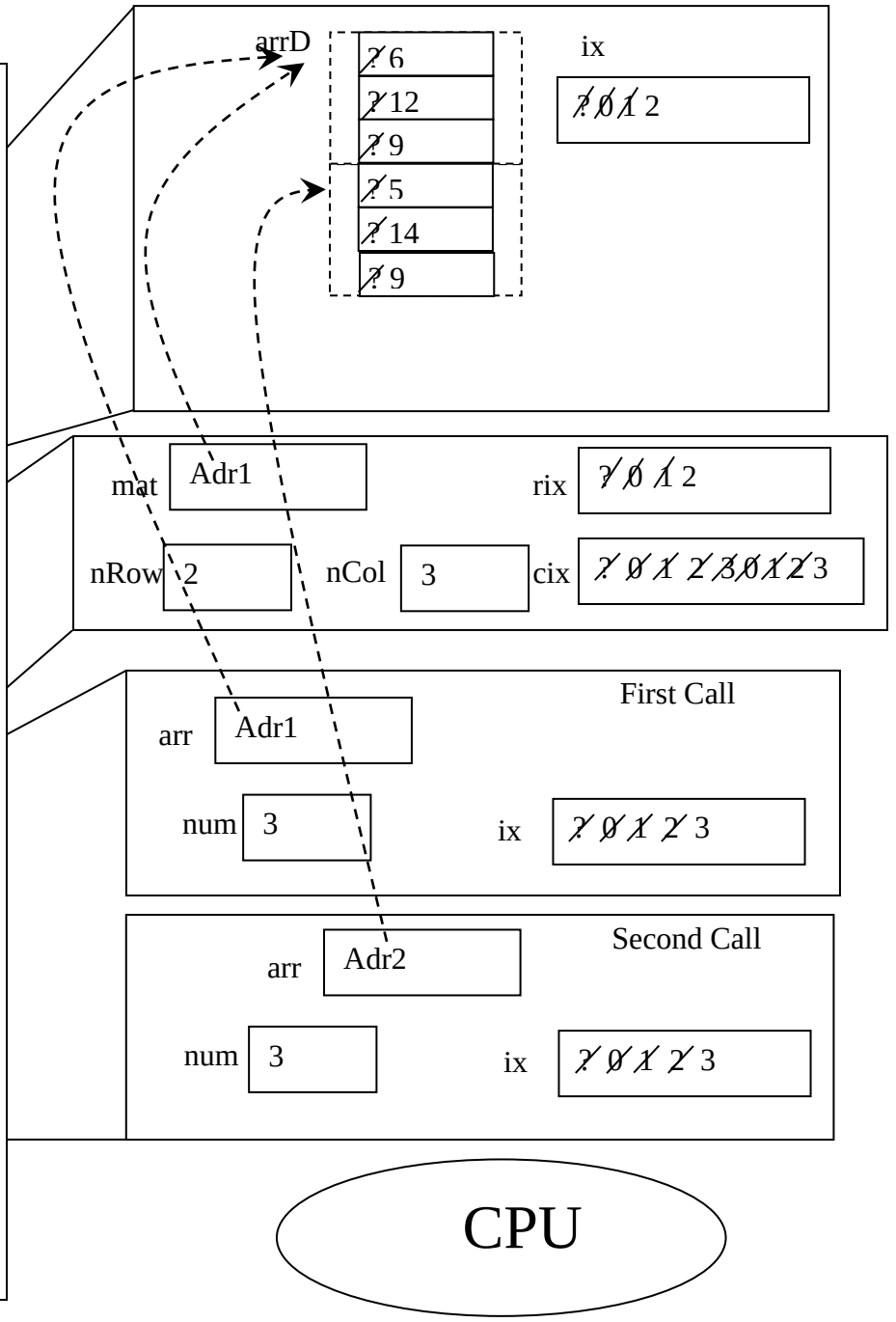
(2)



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



Program Memory

(3)

```
#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

Tues
Fri

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 **Adr1**

size **4**

day **'\0' '\0' '\0' '\0' '\0' '\0' '\0' '\0' '\0' '\0' 'T' 'u' 'e' 's' 'd' 'a' 'y' '\0' '\0' '\0' '\0'**

ptr **Adr2**

index **4**

size **3**

day **'\0' '\0' '\0' '\0' '\0' '\0' '\0' '\0' '\0' '\0' 'F' 'r' 'i' 'd' 'a' 'y' '\0' '\0'**

CPU

Program Memory

(4)

```
#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

months

'J'	'a'	'n'	'u'	'a'	'r'	'y'	'\0'	'\0'	'\0'
'F'	'e'	'b'	'r'	'u'	'a'	'r'	'y'	'\0'	'\0'
'M'	'a'	'r'	'c'	'h'	'\0'	'\0'	'\0'	'\0'	'\0'
'A'	'p'	'r'	'i'	'l'	'\0'	'\0'	'\0'	'\0'	'\0'
'M'	'a'	'y'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'
'J'	'u'	'n'	'e'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'
'J'	'u'	'l'	'y'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'
'A'	'u'	'g'	'u'	's'	't'	'\0'	'\0'	'\0'	'\0'
'S'	'e'	'p'	't'	'e'	'm'	'b'	'e'	'r'	'\0'
'O'	'c'	't'	'o'	'b'	'e'	'r'	'\0'	'\0'	'\0'
'N'	'o'	'v'	'e'	'm'	'b'	'e'	'r'	'\0'	'\0'
'D'	'e'	'c'	'e'	'm'	'b'	'e'	'r'	'\0'	'\0'

ptr **Adr1** size **4**

string

~~'\0'~~	~~'\0'~~	~~'\0'~~	~~'\0'~~	~~'\0'~~	~~'\0'~~	**'\0'**	**'\0'**	**'\0'**	**'\0'**
'A'	**'p'**	**'r'**	**'i'**	**'l'**	**'\0'**				
					'\0'				

ptr **Adr2** start **2** end **7**

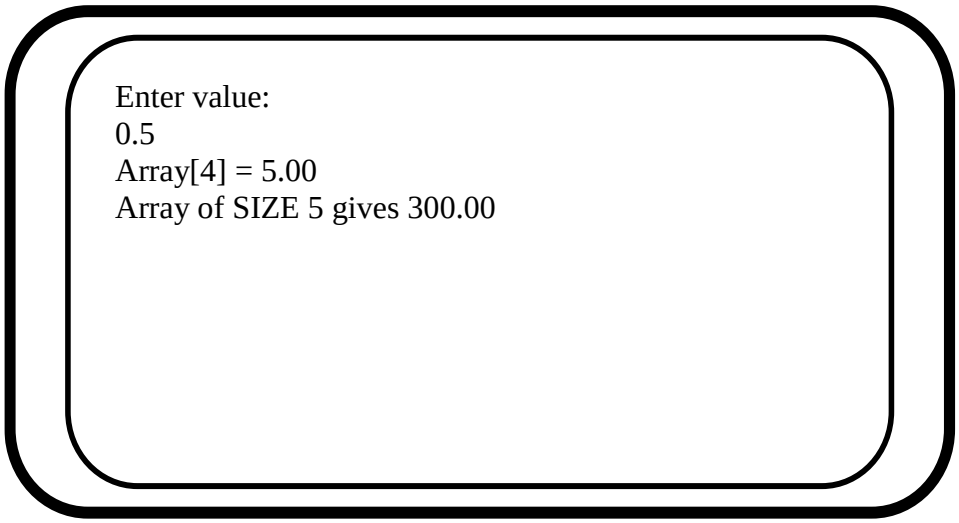
ixString **0 1 2 3 4 5 6** ix **2 3 4 5 6 7 8**

string

'\0'	**'\0'**	**'\0'**	**'\0'**	**'\0'**	**'\0'**	**'\0'**	**'\0'**	**'\0'**	**'\0'**
'p'	**'t'**	**'e'**	**'m'**	**'b'**	**'e'**	**'\0'**			

CPU

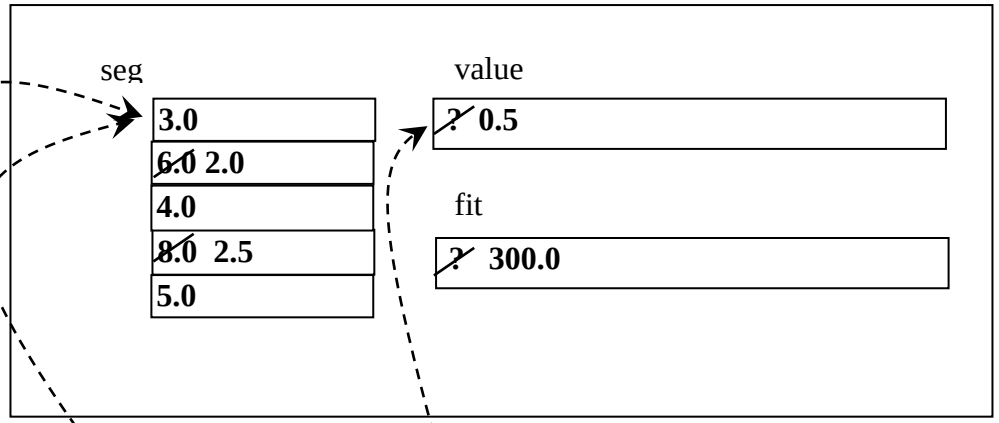
(5)



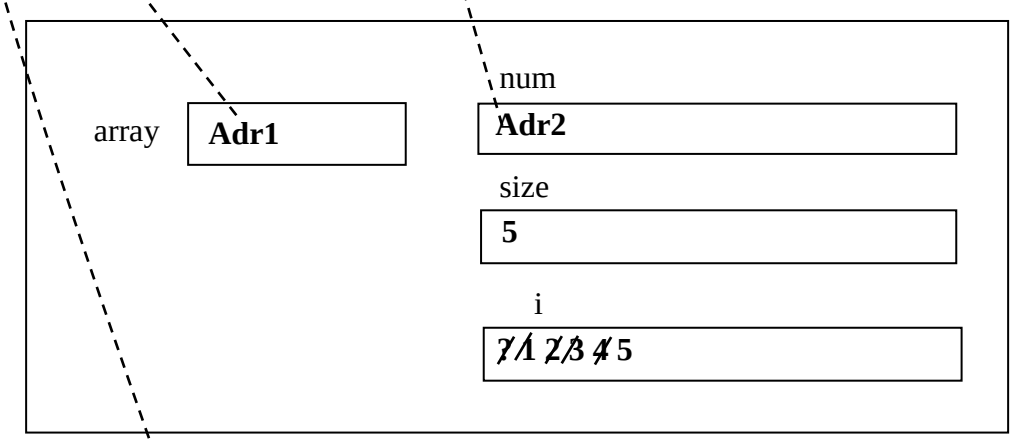
Program Memory

```
#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 %.2f\n",
           SIZE, fit);
}

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

   for(i = 1; i <= size - 1; i = i + 1 )
        if (array[i-1] > array[i])
        {
            array[i-1] = (*num)*array[i];
        }
   printf("Array[%d]=%.2f\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 = i + 1)
   {
        temp=temp*arr[i];
   }
   return(temp);
}
```

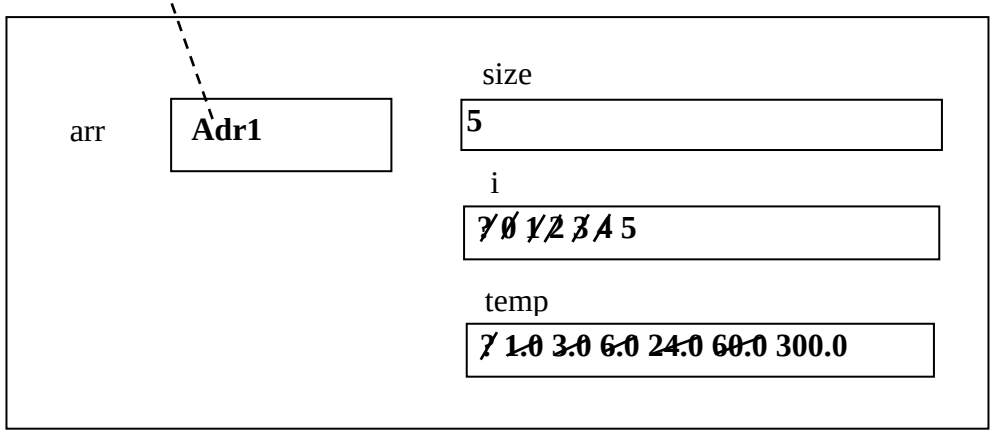
Working Memory (main)



Working Memory (suprog)



Working Memory (compress)



Programming Exercises

1)

(a)

```
/*-------------------------------------------------------------------------
Fonction: 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[])
{
    // variable declarations
    int ix;  // for indexing into the arrays
    double avgTemp;  // for calculating the average temperature
    // Instructions
    // Scan the array and sum all temperature values
    avgTemp = 0.0;
    for(ix = 0; ix < numDays; ix = ix +1)
    {
        avgTemp = avgTemp + temps[ix];
    }
    // Divide by numDays to get average
    avgTemp = avgTemp/numDays;
    return(avgTemp);
}
```

(b)

```
/*-------------------------------------------------------------------------
Fonction: 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 threholdTemp.
Description: Scans the referenced arrays counting the number of temperatures
             whose value exceeded the threshold temperature.
-------------------------------------------------------------------------*/
int numHotDays(double thresholdTemp, int numDays, double temps[])
{
    // variable declarations
    int ix;  // for indexing into the arrays
    int numHotDays;  // for counting number of hot days
    // Instructions
    // Scan the array and count number of hot days
    numHotDays = 0;
    for(ix = 0; ix < numDays; ix = ix +1)
    {
        if(temps[ix] > thresholdTemp)
            numHotDays = numHotDays + 1;
    }
    return(numHotDays);
}
```

(2)

```
/*-------------------------------------------------------------------------
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)
{
    // Variable declations
    double sqrt; // For computing the square root
    double sqrtOld;  // for comparison
    // Instructions
    sqrt = x;
    do
    {
        sqrtOld = sqrt;  // Saves current value for computing difference
        sqrt = (x + sqrtOld*sqrtOld)/(2*sqrtOld);
    } while(fabs(sqrt - sqrtOld) > x/1000.0);
    return(sqrt);
}
```

(3)

```
/*-------------------------------------------------------------------------
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)
{
    // Variable declarations
    double e;
    int i;
    double term;  // for saving value of previous term
    // Instructions
    e = 1.0;
    term = 1; // start at 1
    for(i = 1; i <= n; i = i+1)
    {
        term = term/i;   // promotion of value of i to a double
        e = e + term;
    }
    return(e);
}
```

(4)

```
/*-------------------------------------------------------------------------
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])
{
    int rix;  // row index
    int cix;  // column index
    int isArrHd = TRUE; // assume that it's arrowhead
    // Scan the matrix
    // Note that the value of isArrHd must remain TRUE
    // to continue looping - this makes the function efficient
    for(rix = 0 ; rix < dim && isArrHd ; rix = rix+1)
    {
        for(cix = 0 ; cix < dim && isArrHd ; cix = cix+1)
        {
            if(rix==0 || cix== 0 || rix==cix) // first row/column and diagonal
            {
                if(mat[rix][cix] == 0) isArrHd = FALSE;
            }
            else // zero elements
            {
                if(mat[rix][cix]!=0) isArrHd = FALSE;
            }
        }
    }
    return(isArrHd);
}
```

(5)

```
/*-------------------------------------------------------------------------
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])
{
    int rix;  // row index
    int cix;  // column index
    int flag = TRUE; // assume that it's arrowhead

    for(rix = 0 ; rix < dim && flag ; rix = rix+1)
    {
        if(mat[rix][rix] != 1) flag = FALSE; // diagonal
        for(cix = 0 ; cix < dim && flag ; cix = cix+1)
        {
            if(rix!=cix && mat[rix][cix] != 0) // non-diagonal
                if(mat[rix][cix] !=0 ) flag = FALSE;
        }
    }
    return(flag);
}
```

(6)

(a)

```
/*-------------------------------------------------------------------------
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)
{
    int rix;  // row index
    int cix;  // column index

    // First set the dimensions in the transposed matrix
    matT->nrows = mat->ncols;
    matT->ncols = mat->nrows;

    // Copy values to tranpose matrix
    // Scan each element in mat and
    // save into matT by exchanging rix and cix indexes
    for(rix = 0; rix < mat->nrows; rix = rix + 1)
        for(cix = 0; cix < mat->ncols; cix = cix + 1)
            matT->mat[cix][rix] = mat->mat[rix][cix];
}
```

(b)

```
/*-------------------------------------------------------------------------
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)
{
    int rix;  // row index
    int cix;  // column index

    // Row on one line, align values, and format values
    for(rix = 0; rix < mat->nrows; rix = rix + 1)
    {
        // print all values of same row on a single line
        for(cix = 0; cix < mat->ncols; cix = cix + 1)
            printf("%10.3f ",mat->mat[rix][cix]);
        putchar('\n');  // go to next line
    }
}
```