Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 1 (Compulsory)

(a) Given the following:

```
int x = 5;
printf("%d", ++x);
```

Explain the output. [1]

The output would be 6 as x is incremented by 1 before being printed out. [1]

(b) (i) Define an enumerated type called EmpType that has the following elements: [2]

- Clerk
- Supervisor
- Manager
- Director

```
enum EmpType { Clerk, Supervisor, Manager, Director }

// 1 mark for a correct enumerated structure
// 1 mark for defining the 4 elements in it
```

(ii) What are the ordinal values of Supervisor and Director? [1]

Superviosr = 1, Director = 3
Ensure answer is consistent with solution given in b(i) [1]

(iii) Declare a structure called EmpRec consisting of the following elements: [3]

- EmpCode : 5 characters
- Name : 30 characters
- Type : of EmpType above

```
struct EmpRec
{       char EmpCode[5];
        char Name[50];
        enum EmpType Type ;

}

// 1 mark for a correct EmpRec structure
// 1 mark for defining the 2 character fields correctly
// 1 mark for defining Type correctly
```

(c) Trace the following program and write down its output: [5]

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

char *T1, *T2, *T3, *T4, *T5 ;
char TRACE[] = {'T', 'R', 'A', 'C', 'E'};

main()
{
  clrscr();
  T5 = T4 = TRACE;
  ++T4;
  T3 = &TRACE[4];
  T2 = T5 + 2 ;
  T1 = &TRACE[3] - 3 ;

  printf ("%c\t%c\t%c\t%c\t%c", *T1,*T2,*T3,*T4,*T5);

  getch();
  return 0;
}
```

//T A E R T **[5]**
Marks awarded for partial solutions with correct intermediate steps

(d) Given below:

```
int  OLD = 10;
int  * NEW  ;
NEW = &OLD;
*NEW = OLD * 2;
printf (“%d”,  OLD);
```

State the output. [1]

20; **[1]**

(e) Write a function called ReturnSmallerChars that takes in 2 five-character string parameters str1 and str2. The function compares each of the corresponding characters in str1 and str2 and copies the smaller or equivalent character (with the smaller or equal ASCII code) into a temporary string and then returns it. For example ReturnSmallerChars ("ABC", "abc") returns "ABC" (as 'A' of str1 is smaller than 'a' of str2 in cell 0 and so on). [5]

```
char* ReturnSmallerChars (char str1[], char str2[])
{       char* temp;
        int x=0;
        for (x=0; x<5; x++)
        {       if (str1[x] < str2[x])
                        temp[x] = str1[x];
                else
                        temp[x] = str2[x];
        }
        return temp;
}

// 1 mark for a correct function header
// 1 mark for a correct iteration loop
// 1 mark for comparing str1 and str2
// 1 mark for assigning the str1 characters correctly into the temp string
// 1 mark for assigning the str2 characters correctly into the temp string
```

(f) The function WhatIsWrong given below, takes in an array of floating point numbers and an integer parameter numrec (which stores the number of values in the array). The function WhatIsWrong sums up all values in the array and returns the average. Explain what is wrong with the function. [1]

```
float WhatIsWrong (float floatarray[], int numrec)
{       int i = 0;
        float average = 0;

        while (i < numrec)
        { average += floatarray[i];
        }

        return (average/numrec);
}
```

The value of i is not incremented causing endless looping. [1]

(g) (i) Write a *recursive* function called CountEvenASCII that takes in a string parameter str and returns the number of characters with even ASCII codes. For example

CountEvenASCII("ABCDE") returns 2 (as B and D are of even ASCII codes). [5]

```
int CountEvenASCII (char *str)
{
  if (*str == '\0')
        return 0;
  else
  {
        if ((*str % 2) == 0) // even ASCII code
              return 1 + CountEvenASCII(++str);

        else
              return 0 + CountEvenASCII(++str);
   }
}
```

// 1 mark for a correct function header
// 1 mark for a correct termination case
// 1 mark for checking even ASCII code
// 1 mark for adding 1 and recurring when it is an even ASCII code
// 1 mark for adding 0 and recurring when it is an even ASCII code
Award similar marks for a solution that uses an array string index as opposed to a pointer.

(ii) Change your solution above into an *iterative* solution. [3]

```
int CountEvenASCII (char *str)
{       int count = 0;
        while (*str != '\0')
        {  if ((*str % 2) == 0)
              count++;
           ++str;
        }
        return count;
}
```

// 1 mark for a correct function header
// 1 mark for a correct iteration loop
// 1 mark for checking even ASCII code and increasing count
Award similar marks for a solution that uses an array string index as opposed to a pointer.

(iii) State the main characteristic of a recursive function. [1]

A recursive function must call itself [1]

(h) The function MulBy5 multiplies 5 to the integer parameter passed into it.

Given the 2 function headers below, write the function bodies.

```
void  MulBy5(int *num)
int  MulBy5(int num)
```

[2]

```
void  MulBy5 (int *num)
{         *num = *num * 10; [1]
}

int  MulBy5 (int num)
{         return num * 5; [1]
}
```

Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 2

(a) Explain the purpose of the null character in an array of characters. [1]

The null character indicates the end of the string [1]

(b) Given:

```
char * str1;
char ** str2;
```

Explain the difference between the variables str1 and str2. [2]

Str1 is a string (or a pointer to a character) [1] whereas str2 is a pointer to a string [1]

(c) Write a function called ReturnNextCase that takes in a character parameter ch and returns the next letter in the other case. For example, ChangeCase ('A') returns 'b' and ChangeCase ('a') returns 'B'. You may assume the character parameter ch is always alphabetic. [3]

```
char ReturnNextCase (char ch)
{ if ((ch >= 'A') && (ch<='Z'))
        return (ch + 32 + 1);
  else
        return (ch – 32 + 1);
}

// 1 mark for a correct function header
// 1 mark for checking ch to be either an upper or lower case letter
// 1 mark for the correct conversions
```

(d) Write a function called SortChars that take in two character parameters c1 and c2. If c1 is smaller than c2 then a string with c1 in cell 0 and c2 in cell 1 is returned, otherwise a string with c2 in cell 0 and c1 in cell 1 is returned. For examples, SortChars('3', '2') returns '23' and SortChars ('1', '2') returns '12'. [4]

```
char* SortChars  (char c1, char c2)
{  char temp [2];
   if (c1 <= c2)
  {      temp[0] = c1;
         temp[1] = c2;
   }

   if (c2 <= c1)
   {     temp[0] = c2;
         temp[1] = c1;
   }

   return temp;

}

// 1 mark for a correct function header
// 1 mark for checking c1 to be smaller than c2 and assigning them to
temp
// 1 mark for checking c1 to be bigger than c2 and assigning them to temp
// 1 mark for returning temp
```

(e) Write a function called ReturnASCIIDifference that takes in a string parameter str. The function sums up separately even ASCII characters and odd ASCII characters and returns their difference. For example, ReturnASCIIDifference ('ABCD') returns 2 (as the ASCII values of 'A', 'B', ‘C’ and ‘D’ are 65, 66, 67, 68 respectively and (66+68) - (65+67) = 2). [5]

```
int ReturnASCIIDifference  (char str[])
{        int eventotal = 0, oddtotal = 0; x = 0;
         for (x=0; str[x] != '\0'; x++)
         {   if ((str[x] % 2) == 0)
                   eventotal = eventotal + str[x];
             else
                   oddtotal = oddtotal + str[x];
         }
         return eventotal - oddtotal;
}

// 1 mark for a correct function header
// 1 mark for a correct iteration loop
// 1 mark for summing up even ASCII characters
// 1 mark for summing up odd ASCII characters
// 1 mark for returning the difference
```

Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 3

(a) Explain the difference between *reference* parameters and *value* parameters. [1]

A reference parameter contains the address of the value; whereas a value parameter contains the value itself, be it a data type like int, float, double and char. [1]

(b) Create a *reference* variable num and assign the content value 30 to it. [1]

```
int * num;
*num = 30; [1]
```

(c) Write the function CheckPosNeg that takes in a *reference* integer parameter called num. The function returns 0 if the contents of the reference parameter is a positive number and 1 otherwise. [3]

```
int CheckPosNeg (int * num)
{       if (*num>0)
                return 0;
        else
                return 1;

}

// 1 mark for a correct function header
// 1 mark for de-referencing and checking that num is positive or negative
// 1 mark for returning 1 or 0 correctly
```

(d) Write a function called SumDiff that takes in 2 integer arrays arr1 and arr2 of size 3. The function sums up the absolute difference of each corresponding cell value and returns this sum. For example, if arr1 = {1,2,3} and arr2 = {3,2,1} then the function returns 4 (absolute value of (1-3) + absolute value of (2-2) + absolute value of (3-1)). (Hint: Compare the corresponding values in the arrays and find out which is bigger) [5]

```
int SumDiff (int arr1[10], int arr2[10])
{       int x, diff = 0, sum = 0;
        for (x=0; x<3; x++)
                {       if (arr1[x] > arr2 [x])
                                diff = diff + (arr1[x] – arr2[x]);
                        else
                                diff = diff + (arr2[x] – arr1[x]);

                        sum = sum + diff;
                }

        return sum;
}

// 1 mark for a correct function header
// 1 mark for a correct iteration loop
// 1 mark for checking if arr1 is greater than arr2
// 1 mark for summing up the absolute difference
// 1 mark for returning the sum
```

(e) Write a function called PrintToZero that takes in a reference parameter num. The function prints out all the integers from num down to zero. For example, if *num = 10, then PrintToZero(num) prints 10 9 8 7 .... 0. [5]

```
void PrintToZero (int * num)
{
        if (*num == 0)
                printf("%d", *num);
        else
  {     printf ("%d", *num);
        --(*num);
        PrintToZero (num);
  }
}

// 1 mark for function header
// 1 mark for de-referencing num
// 1 mark for termination condition
// 1 mark for recurring
// 1 mark for decreasing num
```

Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 4

James Timber owns a large piece of forest. He runs a business cutting down the trees and selling them. He would like you to develop a program that tracks the trees in his forest.

(a) State one advantage of arrays. [1]

Arrays provide random access to any cell. [1]

(b) When a tree is cut down it becomes timber and is given an identification code consisting of:

- Zone : a character representing the location zone of the tree
- ID : a 10-character identification number

Define a structure called TimberIDType that contain the above information. [2]

```
struct TimberIDType
{       char Zone;
        char ID [10];
}

// 1mark for the structure
// 1 mark for the 2 fields
```

(c) James would like to store the following information for each of his timber:

- TimberID : of TimberIDType above
- Kind : a 30 character description of the timber
- Diameter : a float number containing the diameter of the timber
- Weight : a float number containing the weight of the timber
- Height : a float number containing the height of the timber

Create a structure called TimberRec that stores the above information. [3]

```
struct TimberRec
{       struct TimberIDType TimberID;
        char Kind[30];
        float Diameter;
        float Weight;
        float Height;

};

// 1 mark for a correct structure
// 1 mark for correct TimberID of TimberIDType
// 1 mark for correct Diameter, Weight and Height
```

(d) Declare an array called TimberArray that can store up to 1,000,000 TimberRec records. [1]

struct TimberRec TimberArray[1000000]; **[1]**

(e) James would like to know the information of timber in a particular zone. Write a function called ShowZoneTimber that takes in the TimberArray array, an integer parameter numrec (which holds the number of timber records) and a character parameter called ReqZone. The function displays information of timber from ReqZone zone. [4]

```
void ShowZoneTimber (struct TimberRec TimberArray[], int numrec, char
ReqZone )
{       int x = 0;
        for (x=0; x<numrec; x++)
                if (ReqZone == TimberArray[x].TimberID.Zone)
                {       printf (“%s”, TimberArray[x].Kind);
                        printf (“%d”, TimberArray[x].Diameter);
                        printf (“%d”, TimberArray[x].Weight);
                        printf (“%d”, TimberArray[x].Height);

                }
}

// 1 mark for a correct function header
// 1 mark for iterating to numrec
// 1 mark for correct checking of the zone
// 1 mark for displaying the fields correctly
```

(f) James wants to know the number of timber that are within a certain weight range. Write a function called DisplayWeightRange that takes in the TimberArray array, an integer parameter numrec which holds the number of timber records, and 2 other float parameters StartWeight and EndWeight that hold the required weight range. [4]

```
void DisplayWeightRange (struct TimberRec TimberArray[], int numrec,
float StartWeight, float EndWeight )
{       int x = 0;
        for (x=0; x<numrec; x++)
                if ((TimberArray[x].Weight >= StartWeight) &&
                    (TimberArray[x].Weight<= EndWeight)
                {       printf (“%s”, TimberArray[x].Kind);
                        printf (“%d”, TimberArray[x].Diameter);
                        printf (“%d”, TimberArray[x].Weight);
                        printf (“%d”, TimberArray[x].Height);

                }
}

// 1 mark for a correct function header
// 1 mark for iterating to numrec
// 1 mark for correct checking the weight range
// 1 mark for displaying the fields correctly
```

Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 5

Fruity Fruit owns a shop selling fruits. He asks you to develop a program to keep track of the fruits in his shop.

(a) Fruity wants to store the following information for each of his fruit:

- BatchID : a 10-character batch id of the fruit
- ThrowDate : the date the fruit expires and needs to be thrown away in ddmmyyyy (you may store it as 8 characters)
- Price : a float containing the selling price of the fruit
- Desc : a 30 -character description of the fruit

Define a structure called FruitRec that consist of the above data. [2]

```
struct FruitRec
{       char BatchID[10];
        char ThrowDate[8];
        float Price;
        char Desc[30];
};

// 1 mark for a correct structure
// 1 mark for the 4 correct fields
```

(b) You have decided to store the following information as a dynamic linked list structure. Create a FruitsList structure containing the following data:

- Fruit : of FruitRec above
- NextFruit : a pointer pointing to the next fruit record [3]

```
struct FruitsList
{       struct FruitRec Fruit;
        struct  FruitsList * NextFruit;
}

// 1 mark for a correct  FruitsList structure
// 1 mark for a correct Fruit field
// 1 mark for NextFruit which points to FruitsList
```

(c) Fruity wants to know all the fruits from a particular batch. Write a function called ShowFruitBatch that takes in Fruits of FruitsList type and a 10-character parameter called ReqBatchID. The function displays all fruits' information belonging to the ReqBatchID. [5]

```
void ShowFruitBatch (struct FruitsList *  Fruits, char ReqBatchID[10])

{        struct FruitsList * temp;
         temp = Fruits;
while (temp != NULL)
{
         if  ((strcmp(temp->BatchID, ReqBatchID) == 0)
         {                 printf (“%s”, temp->ThrowDate);
                           printf (“%f”, temp->Price);
                           printf (“%s”, temp->Desc);
                  }
                  temp = temp->NextFruit;
         }
}
```

// 1 mark for a correct function header
// 1 mark for defining temp to store Fruits
// 1 mark for iterating to the end of list using the NextFlight pointer
// 1 mark for comparing ReqBatchID and temp->BatchID
// 1 mark for displaying the required fields
Award marks for partial solutions which show some of the above points

(d) Fruity wants to know which fruits are going to expire (so they can be thrown away) within the next 3 days. Assuming the current date is 01082003 (in ddmmyyyy), write a function called ShowToBeExpired that takes in Fruits of FruitsList type. The function displays all the information on the fruits that are going to expire within the next 3 days from the current date. [5]

```
void ShowToBeExpired (struct FruitsList *  Fruits)

{       struct FruitsList * temp;
        temp = Fruits;
while (temp != NULL)
{
        if  ((strcmp(temp->ThrowDate, "04082003") < 0) // within next 3 days
        {               printf (“%s”, temp->ThrowDate);
                        printf (“%f”, temp->Price);
                        printf (“%s”, temp->Desc);
                }
                temp = temp->NextFruit;
        }
}
```

// 1 mark for a correct function header
// 1 mark for defining a temp to store Fruits
// 1 mark for iterating to the end of list using the NextFruit pointer
// 1 mark for comparing ThrowDate with "04082003"
// 1 mark for displaying the required fields
Award marks for partial solutions which show some of the above points

- END OF PAPER -