Question Bank: Programming in C - Primitive data types

A. Objective Type Questions: (1 marks)

1. Maximum allowable width of a variable in Turbo C is:

   i) 32 characters ii) 64 characters iii) 128 characters iv) 256 characters

   Answer: i) 32 characters

2. A character variable can at a time store:

   i) 2 characters ii) 3 characters iii) 0 character iv) 1 character

   Answer: d) 1 character

3. Point out which of the following variable names are invalid:

   a) gross-salary b) INTEREST c) salary of emp d) avg. e) thereisagirlinmysoup

   i) a, c & e ii) c, d & e iii) a, c & d iv) none

   Answer: iii) a, c & d

4. Which of the following is not a valid declaration in C?

   a) short int x; b) signed short x; c) short x; d) unsigned short x;

   i) c & d

   ii) b

   iii) a

   iv) all are valid

   Answer: iV) All are valid. First 3 mean the same thing. 4th means unsigned.

5. In C, sizes of a pointer must be same with the following data type:

   i) integer

   ii) float

   iii) long int

   iv) can't say

   Answer: iv) can't say. Sizes of any data type and pointer are compiler dependent. The both sizes need not be same.

6. Suppose a C program has floating constant 1.414, what's the best way to convert this as "float" data type?

   i) (float)1.414

   ii) float(1.414)

   iii) 1.414f or 1.414F

   iV) 1.414 itself of "float" data type i.e. nothing else required.

   Answer: iii) 1.414f or 1.414F. By default floating constant is of double data type. By suffixing it with f or F, it can be converted to float data type.

7. “typedef” in C basically works as an alias. Which of the following is correct for “typedef”?

   i) typedef can be used to alias compound data types such as struct and union.

   ii) typedef can be used to alias both compound data types and pointer to these compound types.

   iii) typedef can be used to alias a function pointer

   iv) typedef can be used to alias an array

   v) All of the above

   Answers: v) All of the above. In C, typedef can be used to alias or make synonyms of other types. Since function pointer is also a type, typedef can be used here. Since, array is also a type of symmetric data types, typedef can be used to alias array as well.

8. what will be the output:

   ```
   #include <stdio.h>

   int main() {

     float c = 5.0;

     printf ("Temperature in Fahrenheit is %.2f", (9/5)*c + 32);

     return 0;

   }
   ```

   i) Temperature in Fahrenheit is 41.00

   ii) Temperature in Fahrenheit is 37.00

   iii) Temperature in Fahrenheit is 0.00

   iv) Compiler Error

   Answer: ii) Temperature in Fahrenheit is 37.00. Since 9 and 5 are integers, integer arithmetic happens in subexpression (9/5) and we get 1 as its value. To fix the above

program, we can use 9.0 instead of 9 or 5.0 instead of 5 so that floating point arithmetic happens.

B. Understanding Questions: (5 marks)

1. Suppose n and p are unsigned int variables in a C program. We wish to set p to ${}^{n}C_{3}$. If n is large, which of the following statements is most likely to set p correctly? Explain
   - p = n * (n-1) * (n-2) / 6;
   - p = n * (n-1) / 2 * (n-2) / 3;
   - p = n * (n-1) / 3 * (n-2) / 2;
   - p = n * (n-1) * (n-2) / 6.0;

   Answer: p = n * (n-1) / 2 * (n-2) / 3; As n is large, the product n*(n-1)*(n-2) will go out of the range(overflow) and it will return a value different from what is expected. Therefore, option (A) and (D) are eliminated. So we consider a shorter product n*(n-1). n*(n-1) is always an even number. So the subexpression " n * (n-1) / 2 " in option B would always produce an integer, which means no precision loss in this subexpression. And when we consider " n*(n-1)/2*(n-2) ", it will always give a number which is a multiple of 3. So dividing it with 3 won't have any loss.

2. **What does the following fragment of C-program print? Explain.**

```
char c[] = "GATE2011";
char *p =c;
printf("%s", p + p[3] - p[1]) ;
```

Answer: The answer is 2011.

```
char c[] = "GATE2011";

// p now has the base address string "GATE2011"
char *p =c;

// p[3] is 'E' and p[1] is 'A'.
// p[3] - p[1] = ASCII value of 'E' - ASCII value of 'A' = 4
// So the expression p + p[3] - p[1] becomes p + 4 which is
// base address of string "2011"
printf("%s", p + p[3] - p[1]) ;
```

3. Find the output with justification:

```
#include <stdio.h>

int main() {

        double x, d=4.4;

         int i=2, y;

        x=(y=d/i)*2;

         printf("x=%lf y=%d\n",x,y);
```

```
    y=(x=d/i)*2;
    printf("x=%lf y=%d\n",x,y);
    return 0;
}
```

Answer:

Output

```
x = 4.000000 y = 2
x = 2.200000 y = 4
```

Explanation

Let us analyse the first expression step by step after replacing the values of the variables in the expression.

```
x = ( y = 4.4 / 2 ) * 2
x = ( y = 2.2 ) * 2
x = 2 * 2
x = 4
```

Note that 2.2 is demoted to 2 while storing it in **y**, since **y** is an integer variable and hence can store only an integer constant.

Onto the second expression now.

```
y = ( x = d / i ) * 2
y = ( x = 2.2 ) * 2
y = 2.2 * 2
y = 4.4
```

While storing 4.4 in **y**, it is demoted to 4, since **y** is an integer variable.

4. Find the output of the following program with explanation.

```
#include <stdio.h>
    int i;
    int main() {
        int j;
        for(;;)
            {
                if (j=function(i))
                    printf ("j=%d\n",j);
```

```
                    else break;
                }
        return 0;
        }
function (x)
int x;
        {
                static int v=2;
                v--;
                return (v-x);
        }
```

Answer:

Output

j = 1

Explanation

In this program, having declared **i** as an **extern** variable, the control enters the **for** loop in **main( )**. Here **function( )** is called with a value 0. In **function( )** this value is collected in the variable **x**. Next **v** is initiated to 2, decremented to 1 and then the difference (**v - x**), i.e. 1, is returned to **main( )**, where it is collected in the variable **j**. Since **j** is 1, the condition is satisfied and **j**'s value gets printed. Since the **for** loop is an indefinite loop, once again the control reaches the **if** statement. From here, again a call to **function( )** is made, with **i** equal to 0. In **function( )** the latest value of **v**, i.e. 1 is decremented to 0 and the difference (**0 - 0**) is returned to **main( )**, where it is collected in **j**. Now the condition fails, since **j** is 0, and the control reaches the **else** block. Here the **break** takes the control outside the **for** loop and since there are no statements outside the loop, the execution is terminated.

C. Scenario Based Questions: (10 marks)

1. Write a C program to find the median of an unsorted array. If your solution runs in $O(n^2)$ time can there be any approach to find it in linear time or the time better than $O(n^2)$? (8+2=10)

   [Note: Median of a sorted array of size N is defined as the middle element when n is odd and average of middle two elements when n is even.]

   Answer:

```
#include<stdio.h>
#define N 10
int main( ){
  int i,j,n;
  float median,a[N],t;
   printf("Enter the number of items");
   scanf("%d", &n);
   /* Reading items into array a */
   printf("Input %d values",n);
   for (i = 1; i <= n ; i++)
   scanf("%f", &a[i]);
 /* Sorting begins */
  for (i = 1 ; i <= n-1 ; i++){ /* Trip-i begins */
    for (j = 1 ; j <= n-i ; j++) {
      if (a[j] <= a[j+1]) { /* Interchanging values */
       t = a[j];
       a[j] = a[j+1];
       a[j+1] = t;
   }
   else
    continue ;
      }
   } /* sorting ends */
   /* calculation of median */
   if ( n % 2 == 0)
     median = (a[n/2] + a[n/2+1])/2.0 ;
   else
   median = a[n/2 + 1];
```

```
  /* Printing */
 for (i = 1 ; i <= n ; i++)
  printf("%f ", a[i]);
  printf("Median is %f", median);
  return 0;
}
```

Approach to find in linear time:

Our next step will be to *usually* find the median within linear time, assuming we don't get unlucky. This algorithm, called "quickselect", was devevloped by Tony Hoare who also invented the similarly-named quicksort. It's a recursive algorithm that can find any element (not just the median).

1. **Pick an index in the list. It doesn't matter how you pick it, but choosing one at random works well in practice. The element at this index is called the pivot.**
2. **Split the list into 2 groups:**
    1. **Elements less than or equal to the pivot, lesser_els**
    2. **Elements strictly greater than the pivot, great_els**
3. **We know that one of these groups contains the median. Suppose we're looking for the *kth* element:**

    - **If there are *k* or more elements in lesser_els, recurse on list lesser_els, searching for the *kth* element.**
    - **If there are fewer than *k* elements in lesser_els, recurse on list greater_els. Instead of searching for *k*, we search for k-len(lesser_els).**

2. Given an unsorted array that contains even number of occurrences for all numbers except two numbers. Write a program to find the two numbers which have odd occurrences in O(n) time complexity and O(1) extra space.

   [Hint: Use XOR operation]

   Examples:

   Input: {12, 23, 34, 12, 12, 23, 12, 45}

   Output: 34 and 45

   Input: {4, 4, 100, 5000, 4, 4, 4, 4, 100, 100}

   Output: 100 and 5000

   Input: {10, 20}

   Output: 10 and 20

   Answer:

   // Program to find the two odd occurring elements

   #include<stdio.h>

```
/* Prints two numbers that occur odd number of times. The
   function assumes that the array size is at least 2 and
   there are exactly two numbers occurring odd number of times. */
void printTwoOdd(int arr[], int size)
{
 int xor2 = arr[0]; /* Will hold XOR of two odd occurring elements */
 int set_bit_no;  /* Will have only single set bit of xor2 */
 int i;
 int n = size - 2;
 int x = 0, y = 0;

 /* Get the xor of all elements in arr[]. The xor will basically
    be xor of two odd occurring elements */
  for(i = 1; i < size; i++)
   xor2 = xor2 ^ arr[i];

  /* Get one set bit in the xor2. We get rightmost set bit
    in the following line as it is easy to get */
  set_bit_no = xor2 & ~(xor2-1);

  /* Now divide elements in two sets:
   1) The elements having the corresponding bit as 1.
   2) The elements having the corresponding bit as 0.  */
  for(i = 0; i < size; i++)
  {
     /* XOR of first set is finally going to hold one odd
        occurring number x */
    if(arr[i] & set_bit_no)
     x = x ^ arr[i];

    /* XOR of second set is finally going to hold the other
       odd occurring number y */
    else
      y = y ^ arr[i];
  }

  printf("\n The two ODD elements are %d & %d ", x, y);
}

/* Driver program to test above function */
int main()
{
  int arr[] = {4, 2, 4, 5, 2, 3, 3, 1};
  int arr_size = sizeof(arr)/sizeof(arr[0]);
  printTwoOdd(arr, arr_size);
  getchar();
  return 0;
}
```