SOLUTIONS OF C LANGUAGE – III

1. **If we have declared an array as global in one file and we are using it in another file then why doesn't the sizeof operator works on an extern array?**

   Ans: An extern array is of incomplete type as it does not contain the size. Hence we cannot use sizeof operator, as it cannot get the size of the array declared in another file. To resolve this use any of one the following two solutions:
   1. In the same file declare one more variable that holds the size of array. For example,

   array.c

   ```
   int arr[5] ;
   int arrsz = sizeof ( arr ) ;
   ```

   myprog.c

   ```
   extern int arr[] ;
   extern int arrsz ;
   ```

   2. Define a macro which can be used in an array declaration. For example,

   myheader.h

   ```
   #define SZ 5
   ```

   array.c

   ```
   #include "myheader.h"
   int arr[SZ] ;
   ```

   myprog.c

   ```
   #include "myheader.h"
   extern int arr[SZ] ;
   ```

   ---

2. **How do I write printf( ) so that the width of a field can be specified at runtime?**

Ans: This is shown in following code snippet.

```
main( )
{
int w, no ;
printf ( "Enter number and the width for the
number field:" ) ;
scanf ( "%d%d", &no, &w ) ;
printf ( "%*d", w, no ) ;
}
```

Here, an '*' in the format specifier in printf( ) indicates that an int value from the argument list should be used for the field width.

---

3. **How to find the row and column dimension of a given 2-D array?**

Ans: Whenever we initialize a 2-D array at the same place where it has been declared, it is not necessary to mention the row dimension of an array. The row and column dimensions of such an array can be determined programmatically as shown in following program.

```
void main( )
{
int a[][3] = { 0, 1, 2,
9,-6, 8,
7, 5, 44,
23, 11,15 } ;

int c = sizeof ( a[0] ) / sizeof ( int ) ;
int r = ( sizeof ( a ) / sizeof ( int ) ) / c ;
int i, j ;

printf ( "\nRow: %d\nCol: %d\n", r, c ) ;
for ( i = 0 ; i < r ; i++ )
{
for ( j = 0 ; j < c ; j++ )
printf ( "%d ", a[i][j] ) ;
printf ( "\n" ) ;
}
}
```

---

4. **The access( ) function...**

The access( ) function checks for the existence of a file and also determines whether it can be read, written to or executed. This function takes two arguments the filename and an integer indicating the access mode. The values 6, 4, 2, and 1 checks for read/write, read, write and execute permission of a given file, whereas value 0 checks whether the file exists or not. Following program demonstrates how we can use access( ) function to check if a given file exists.

```
#include <io.h>

main( )
{
char fname[67] ;

printf ( "\nEnter name of file to open" ) ;
gets ( fname ) ;

if ( access ( fname, 0 ) != 0 )
{
printf ( "\nFile does not exist." ) ;
return ;
}
}
```

---

5. **How do I convert a floating-point number to a string?**

Ans: Use function gcvt( ) to convert a floating-point number to a string. Following program demonstrates the use of this function.

```
#include <stdlib.h>

main( )
{
char str[25] ;
float no ;
int dg = 5 ; /* significant digits */

no = 14.3216 ;
gcvt ( no, dg, str ) ;
```

```
printf ( "String: %s\n", str ) ;
}
```

---

6. What is a stack ?

Ans: The stack is a region of memory within which our programs temporarily store data as they execute. For example, when a program passes parameters to functions, C places the parameters on the stack. When the function completes, C removes the items from the stack. Similarly, when a function declares local variables, C stores the variable's values on the stack during the function's execution. Depending on the program's use of functions and parameters, the amount of stack space that a program requires will differ.

---

7. How to distinguish between a binary tree and a tree?

Ans: A node in a tree can have any number of branches. While a binary tree is a tree structure in which any node can have at most two branches. For binary trees we distinguish between the subtree on the left and subtree on the right, whereas for trees the order of the subtrees is irrelevant.
Consider the following figure...

This above figure shows two binary trees, but these binary trees are different. The first has an empty right subtree while the second has an empty left subtree. If the above are regarded as trees (not the binary trees), then they are same despite the fact that they are drawn differently. Also, an empty binary tree can exist, but there is no tree having zero nodes.

---

8. How do I use the function ldexp( ) in a program?

Ans: The math function ldexp( ) is used while solving the complex mathematical equations. This function takes two arguments, a double value and an int respectively. The order in which ldexp( ) function performs calculations is (n * pow (2, exp)) where n is the double value and exp is the integer. The following program demonstrates the use of this function.

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

void main( )
{
double ans ;
```

```
double n = 4 ;

ans = ldexp ( n, 2 ) ;
printf ( "\nThe ldexp value is : %lf\n", ans ) ;
}
```

Here, ldexp( ) function would get expanded as (4 * 2 * 2), and the output would be the ldexp value is : 16.000000

---

9. Can we get the mantissa and exponent form of a given number?

Ans: The function frexp( ) splits the given number into a mantissa and exponent form. The function takes two arguments, the number to be converted as a double value and an int to store the exponent form. The function returns the mantissa part as a double value. Following example demonstrates the use of this function.

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

void main( )
{
double mantissa, number ;
int exponent ;

number = 8.0 ;
mantissa = frexp ( number, &exponent ) ;

printf ( "The number %lf is ", number ) ;
printf ( "%lf times two to the ", mantissa ) ;
printf ( "power of %d\n", exponent ) ;

return 0 ;
}
```

---

10. How do I write code that executes certain function only at program termination?

Ans: Use atexit( ) function as shown in following program.

```
#include <stdlib.h>
main( )
{
int ch ;
```

```
void fun ( void ) ;
atexit ( fun ) ;
// code
}
void fun( void )
{
printf ( "\nTerminate program....." ) ;
getch( ) ;
}
```

---

11. What are memory models?

Ans: The compiler uses a memory model to determine how much memory is allocated to the program. The PC divides memory into blocks called segments of size 64 KB. Usually, program uses one segment for code and a second segment for data. A memory model defines the number of segments the compiler can use for each. It is important to know which memory model can be used for a program. If we use wrong memory model, the program might not have enough memory to execute. The problem can be solved using larger memory model. However, larger the memory model, slower is your program execution. So we must choose the smallest memory model that satisfies our program needs. Most of the compilers support memory models like tiny, small, medium, compact, large and huge.

---

12. How does C compiler store elements in a multi-dimensional array?

Ans: The compiler maps multi-dimensional arrays in two ways—Row major order and Column order. When the compiler places elements in columns of an array first then it is called column-major order. When the compiler places elements in rows of an array first then it is called row-major order. C compilers store multidimensional arrays in row-major order. For example, if there is a multi-dimensional array a[2][3], then according row-major order, the elements would get stored in memory following order:
a[0][0], a[0][1], a[0][2], a[1][0], a[1][1], a[1][2]

---

13. If the result of an _expression has to be stored to one of two variables, depending on a condition, can we use conditional operators as shown below?

((i < 10) ? j : k) = l * 2 + p ;

Ans: No! The above statement is invalid. We cannot use the conditional operators in this fashion. The conditional operators like most operators, yields a value, and we cannot assign the value of an _expression to a value. However, we can use conditional operators as shown in

following code snippet.

```
main( )
{
int i, j, k, l ;
i = 5 ; j = 10 ; k = 12, l = 1 ;
* ( ( i < 10 ) ? &j : &k ) = l * 2 + 14 ;
printf ( "i = %d j = %d k = %d l = %d", i, j, k, l ) ;
}
```

The output of the above program would be as given below:

```
i = 5 j = 16 k = 12 l = 1
```

---

14. How can I find the day of the week of a given date?

Ans: The following code snippet shows how to get the day of week from the given date.

```
dayofweek ( int yy, int mm, int dd )
{
/*Monday = 1 and Sunday = 0 */
/* month number >= 1 and <= 12, yy > 1752 or so */
static int arr[ ] = { 0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4 } ;
yy = yy - mm < 3 ;
return ( yy + yy / 4 - yy / 100 + yy / 400 + arr[ mm - 1] + dd ) % 7 ;
}

void main( )
{
printf ( "\n\n\nDay of week : %d ", dayofweek ( 2002, 5, 18 ) ) ;
}
```

---

15. What's the difference between these two declarations?

```
struct str1 { ... } ;
typedef struct { ... } str2 ;
```

Ans : The first form declares a structure tag whereas the second declares a typedef. The main difference is that the second declaration is of a slightly more abstract type -- its users don't necessarily know that it is a structure, and the keyword struct is not used when declaring instances of it.
