SOLUTIONS OF C LANGUAGE - II

1. **How do I write code to find an amount of free disk space available on current drive?**

   Ans: Use getdfree( ) function as shown in follow code.

```
#include <stdio.h>
#include <stdlib.h>
#include <dir.h>
#include <dos.h>

main( )
{
int dr ; struct dfree disk ;
long freesp ;

dr = getdisk( ) ;
getdfree ( dr + 1 , &disk ) ;

if ( disk.df_sclus == 0xFFFF )
{
printf ( "\ngetdfree( ) function failed\n");
exit ( 1 ) ;
}

freesp = ( long ) disk.df_avail
* ( long ) disk.df_bsec
* ( long ) disk.df_sclus ;
printf ( "\nThe current drive %c: has %ld bytes
available as free space\n", 'A' + dr, freesp ) ;
}
```

2. Use of array indices...
   If we wish to store a character in a char variable ch and the character to be stored depends on the value of another variable say color (of type int), then the code would be as shown below:

```
switch ( color )
{
case 0 :
ch = 'R' ;
```

```
break ;
case 1 :
ch = 'G' ;
break ;
case 2 :
ch = 'B' ;
break ;
}
```

In place of switch-case we can make use of the value in color as an index for a character array. How to do this is shown in following code snippet.

```
char *str = "RGB' ;
char ch ;
int color ;
// code
ch = str[ color ] ;
```

---

3. Function atexit( ) recevies parameter as the address of function of the type void fun (void). The function whose address is passed to atexit( ) gets called before the termination of program. If atexit( ) is called for more than one function then the functions are called in "first in last out" order. You can verify that from the output.

```
#include <stdio.h>
#include <stdlib.h>
void fun1( )
{
printf("Inside fun1\n");
}

void fun2( )
{
printf("Inside fun2\n");
}
main( )
{
atexit ( fun1 ) ;
/* some code */
atexit ( fun2 ) ;
printf ( "This is the last statement of
program?\n" );
}
```

---

4. **How do I write a user-defined function, which deletes each character in a string str1, which matches any character in string str2?**

   Ans: The function is as shown below:

```
Compress ( char str1[], char str2[] )
{
int i, j, k ;

for ( i = k = 0 ; str1[i] != '\0' ; i++ )
{
for ( j = 0 ; str2[j] != '\0' && str2[j] !=
str1[i] ; j++ )
;
if ( str2[j] == '\0' )
str1[k++] = str1[I] ;
}
str1[k] = '\0'
}
```

   ---

5. **How does free( ) know how many bytes to free?**

   Ans: The malloc( ) / free( ) implementation remembers the size of each block allocated and returned, so it is not necessary to remind it of the size when freeing.

   ---

6. **What is the use of randomize( ) and srand( ) function?**

   Ans: While generating random numbers in a program, sometimes we require to control the series of numbers that random number generator creates. The process of assigning the random number generators starting number is called seeding the generator. The randomize( ) and srand( ) functions are used to seed the random number generators. The randomize( ) function uses PC's clock to produce a random seed, whereas the srand( ) function allows us to specify the random number generator's starting value.

   ---

7. **How do I determine amount of memory currently available for allocating?**

   Ans: We can use function coreleft( ) to get the amount of memory available for allocation. However, this function does not give an exact amount of unused memory. If, we are using a small memory model, coreleft( ) returns the amount of unused memory between the top of the heap and stack. If we are using a larger model, this function returns the amount of memory between the highest allocated memory and the end of conventional memory. The function returns amount of memory in terms of bytes.

   ---

8. **How does a C program come to know about command line arguments?**

   Ans: When we execute our C program, operating system loads the program into memory. In case of DOS, it first loads 256 bytes into memory, called program segment prefix. This contains file table, environment segment, and command line information. When we compile the C program the compiler inserts additional code that parses the command, assigning it to the argv array, making the arguments easily accessible within our C program.

---

9. **When we open a file, how does functions like fread( )/fwrite( ), etc. get to know from where to read or to write the data?**

   Ans: When we open a file for read/write operation using function like fopen( ), it returns a pointer to the structure of type FILE. This structure stores the file pointer called position pointer, which keeps track of current location within the file. On opening file for read/write operation, the file pointer is set to the start of the file. Each time we read/write a character, the position pointer advances one character. If we read one line of text at a step from the file, then file pointer advances to the start of the next line. If the file is opened in append mode, the file pointer is placed at the very end of the file. Using fseek( ) function we can set the file pointer to some other place within the file.

   ---

10. **The sizeof( ) function doesn't return the size of the block of memory pointed to by a pointer. Why?**

    Ans: The sizeof( ) operator does not know that malloc( ) has been used to allocate a pointer. sizeof( ) gives us the size of pointer itself. There is no handy way to find out the size of a block

allocated by malloc( ).

---

11. **FP_SEG And FP_OFF…**

Sometimes while working with far pointers we need to break a far address into its segment and offset. In such situations we can use FP_SEG and FP_OFF macros. Following program illustrates the use of these two macros.

```
#include <dos.h>

main( )
{
unsigned s, o ;
char far *ptr = "Hello!" ;

s = FP_SEG ( ptr ) ;
o = FP_OFF ( ptr ) ;
printf ( "\n%u %u", s, o ) ;
}
```

---

12. **How do I write a program to convert a string containing number in a hexadecimal form to its equivalent decimal?**

Ans: The following program demonstrates this:

```
main( )
{
char str[] = "0AB" ;
int h, hex, i, n ;
n = 0 ; h = 1 ;
for ( i = 0 ; h == 1 ; i++ )
{
if ( str[i] >= '0' && str[i] <= '9' )
hex = str[i] - '0' ;
else
{
if ( str[i] >= 'a' && str[i] <= 'f' )
hex = str[i] - 'a' + 10 ;
else
if ( str[i] >= 'A' && str[i] <= 'F' )
hex = str[i] - 'A' + 10 ;
else
h = 0 ;
```

```
}
if ( h == 1 )
n = 16 * n + hex ;
}
printf ( "\nThe decimal equivalent of %s is %d",
str, n ) ;
}
```

The output of this program would be the decimal equivalent of 0AB is 171.

---

13. **How do I write code that reads the segment register settings?**

Ans: We can use segread( ) function to read segment register settings. There are four segment registers—code segment, data segment, stack segment and extra segment. Sometimes when we use DOS and BIOS services in a program we need to know the segment register's value. In such a situation we can use segread( ) function. The following program illustrates the use of this function.

```
#include <dos.h>
main( )
{
struct SREGS s ;
segread ( &s ) ;
printf ( "\nCS: %X DS: %X SS: %X ES: %X",s.cs,
s.ds, s.ss, s.es ) ;
}
```

---

14. **What is environment and how do I get environment for a specific entry?**

Ans: While working in DOS, it stores information in a memory region called environment. In this region we can place configuration settings such as command path, system prompt, etc. Sometimes in a program we need to access the information contained in environment. The function getenv( ) can be used when we want to access environment for a specific entry. Following program demonstrates the use of this function.

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

main( )
{
char *path = NULL ;

path = getenv ( "PATH" ) ;
```

```
if ( *path != NULL )
printf ( "\nPath: %s", path ) ;
else
printf ( "\nPath is not set" ) ;
}
```

---

15. **How do I display current date in the format given below?**

Saturday October 12, 2002
Ans: Following program illustrates how we can display date in above given format.

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

main( )
{
struct tm *curtime ;
time_t dtime ;

char str[30] ;

time ( &dtime ) ;
curtime = localtime ( &dtime ) ;
strftime ( str, 30, "%A %B %d, %Y", curtime ) ;

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

Here we have called time( ) function which returns current time. This time is returned in terms of seconds, elapsed since 00:00:00 GMT, January 1, 1970. To extract the week day, day of month, etc. from this value we need to break down the value to a tm structure. This is done by the function localtime( ). Then we have called strftime( ) function to format the time and store it in a string str.