SOLUTIONS OF C LANGUAGE - I

1.

What will be the output of the following code?

```
void main ()
{ int i = 0 , a[3] ;
a[i] = i++;
printf (“%d",a[i]) ;
}
```

Ans: The output for the above code would be a garbage value. In the statement a[i] = i++; the value of the variable i would get assigned first to a[i] i.e. a[0] and then the value of i would get incremented by 1. Since a[i] i.e. a[1] has not been initialized, a[i] will have a garbage value.

2.

Why doesn't the following code give the desired result?

```
int x = 3000, y = 2000 ;
long int z = x * y ;
```

Ans: Here the multiplication is carried out between two ints x and y, and the result that would overflow would be truncated before being assigned to the variable z of type long int. However, to get the correct output, we should use an explicit cast to force long arithmetic as shown below:

```
long int z = ( long int ) x * y ;
```

Note that (long int)(x * y) would not give the desired effect.

3.

Why doesn't the following statement work?

```
char str[ ] = "Hello" ;
strcat ( str, '!' ) ;
```

Ans: The string function strcat( ) concatenates strings and not a character. The basic difference between a string and a character is that a string is a collection of characters, represented by an array of characters whereas a character is a single character. To make the above statement work writes the statement as shown below:

```
strcat ( str, "!" ) ;
```

4.

How do I know how many elements an array can hold?

Ans: The amount of memory an array can consume depends on the data type of an array. In DOS environment, the amount of memory an array can consume depends on the current memory model (i.e. Tiny, Small, Large, Huge, etc.). In general an array cannot consume more than 64 kb. Consider following program, which shows the maximum number of elements an array of type int, float and char can have in case of Small memory model.

```
main( )
{
int i[32767] ;
float f[16383] ;
char s[65535] ;
}
```

5.

How do I write code that reads data at memory location specified by segment and offset?

Ans: Use peekb( ) function. This function returns byte(s) read from specific segment and offset locations in memory. The following program illustrates use of this function. In this program from VDU memory we have read characters and its attributes of the first row. The information stored in file is then further read and displayed using peek( ) function.

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

main( )
{

char far *scr = 0xB8000000 ;
FILE *fp ;
int offset ;
char ch ;

if ( ( fp = fopen ( "scr.dat", "wb" ) ) == NULL )
{
```

```
printf ( "\nUnable to open file" ) ;
exit( ) ;

}

// reads and writes to file
for ( offset = 0 ; offset < 160 ; offset++ )
fprintf ( fp, "%c", peekb ( scr, offset ) ) ;
fclose ( fp ) ;

if ( ( fp = fopen ( "scr.dat", "rb" ) ) == NULL )
{

printf ( "\nUnable to open file" ) ;
exit( ) ;

}

// reads and writes to file
for ( offset = 0 ; offset < 160 ; offset++ )
{

fscanf ( fp, "%c", &ch ) ;
printf ( "%c", ch ) ;

}

fclose ( fp ) ;

}
```

6.

How do I compare character data stored at two different memory locations?

Ans: Sometimes in a program we require to compare memory ranges containing strings. In such a situation we can use functions like memcmp( ) or memicmp( ). The basic difference between two functions is that memcmp( ) does a case-sensitive comparison whereas memicmp( ) ignores case of characters. Following program illustrates the use of both the functions.

```
#include <mem.h>
```

```
main( )
{
char *arr1 = "Kicit" ;
char *arr2 = "kicitNagpur" ;

int c ;

c = memcmp ( arr1, arr2, sizeof ( arr1 ) ) ;

if ( c == 0 )
printf ( "\nStrings arr1 and arr2 compared using memcmp are identical" ) ;

else
printf ( "\nStrings arr1 and arr2 compared using memcmp are not identical"
) ;

c = memicmp ( arr1, arr2, sizeof ( arr1 ) ) ;

if ( c == 0 )
printf ( "\nStrings arr1 and arr2 compared using memicmp are identical" )
;
else
printf ( "\nStrings arr1 and arr2 compared using memicmp are not
identical" ) ;
}
```

7.

Fixed-size objects are more appropriate as compared to variable size data objects. Using variable-size data objects saves very little space. Variable size data objects usually have some overhead. Manipulation of fixed-size data objects is usually faster and easier. Use fixed size when maximum size is clearly bounded and close to average. And use variable-size data objects when a few of the data items are bigger than the average size. For example,

```
char *num[10] = { "One", "Two", "Three", "Four",
"Five", "Six", "Seven", "Eight", "Nine", "Ten" } ;
```

Instead of using the above, use

```
char num[10][6] = { "One", "Two", "Three", "Four",
"Five", "Six", "Seven", "Eight", "Nine", "Ten" } ;
```

The first form uses variable-size data objects. It allocates 10 pointers, which are pointing to 10 string constants of variable size. Assuming each pointer is of 4 bytes, it requires 90 bytes. On the other hand, the second form uses fixed size data objects. It allocates 10 arrays of 6 characters each. It requires only 60 bytes of space. So, the variable-size in this case does not offer any advantage over fixed size.

8.

The Spawnl( ) function...

DOS is a single tasking operating system, thus only one program runs at a time. The Spawnl( ) function provides us with the capability of starting the execution of one program from within another program. The first program is called the parent process and the second program that gets called from within the first program is called a child process. Once the second program starts execution, the first is put on hold until the second program completes execution. The first program is then restarted. The following program demonstrates use of spawnl( ) function.

```
/* Mult.c */

int main ( int argc, char* argv[ ] )
{
int a[3], i, ret ;
if ( argc < 3 || argc > 3 )
{
printf ( "Too many or Too few arguments..." ) ;
exit ( 0 ) ;
}

for ( i = 1 ; i < argc ; i++ )
a[i] = atoi ( argv[i] ) ;
ret = a[1] * a[2] ;
return ret ;
}

/* Spawn.c */
#include <process.h>
#include <stdio.h>

main( )
{
int val ;
val = spawnl ( P_WAIT, "C:\\Mult.exe", "3", "10",
"20", NULL ) ;
```

```
printf ( "\nReturned value is: %d", val ) ;
}
```

Here, there are two programs. The program 'Mult.exe' works as a child process whereas 'Spawn.exe' works as a parent process. On execution of 'Spawn.exe' it invokes 'Mult.exe' and passes the command-line arguments to it. 'Mult.exe' in turn on execution, calculates the product of 10 and 20 and returns the value to val in 'Spawn.exe'. In our call to spawnl( ) function, we have passed 6 parameters, P_WAIT as the mode of execution, path of '.exe' file to run as child process, total number of arguments to be passed to the child process, list of command line arguments and NULL. P_WAIT will cause our application to freeze execution until the child process has completed its execution. This parameter needs to be passed as the default parameter if you are working under DOS. under other operating systems that support multitasking, this parameter can be P_NOWAIT or P_OVERLAY. P_NOWAIT will cause the parent process to execute along with the child process, P_OVERLAY will load the child process on top of the parent process in the memory.

9.

Are the following two statements identical?

```
char str[6] = "Kicit" ;
char *str = "Kicit" ;
```

Ans: No! Arrays are not pointers. An array is a single, pre-allocated chunk of contiguous elements (all of the same type), fixed in size and location. A pointer on the other hand, is a reference to any data element (of a particular type) located anywhere. A pointer must be assigned to point to space allocated elsewhere, but it can be reassigned any time. The array declaration char str[6] ; requests that space for 6 characters be set aside, to be known
by name str. In other words there is a location named str at which six characters are stored. The pointer declaration char *str ; on the other hand, requests a place that holds a pointer, to be known by the name str. This pointer can point almost anywhere to any char, to any contiguous array of chars, or nowhere.

10.

Is the following code fragment correct?

```
const int x = 10 ;
int arr[x] ;
```

Ans: No! Here, the variable x is first declared as an int so memory is reserved for it. Then it is qualified by a const qualifier. Hence, const qualified object is not a constant fully. It is an object with read only attribute, and in C, an object associated with memory cannot be used in array dimensions.

11.

How do I write code to retrieve current date and time from the system and display it as a string?

Ans: Use time( ) function to get current date and time and then ctime( ) function to display it as a string. This is shown in following code snippet.

```
#include <sys\types.h>

void main( )
{
time_t curtime ;
char ctm[50] ;

time ( &curtime ) ; //retrieves current time &
stores in curtime
printf ( "\nCurrent Date & Time: %s", ctime (
&curtime ) ) ;
}
```

12.

How do I change the type of cursor and hide a cursor?

Ans: We can change the cursor type by using function _setcursortype( ). This function can change the cursor type to solid cursor and can even hide a cursor. Following code shows how to change the cursor type and hide cursor.

```
#include <conio.h>
main( )
{
/* Hide cursor */
_setcursortype ( _NOCURSOR ) ;

/* Change cursor to a solid cursor */
_setcursortype ( _SOLIDCURSOR ) ;

/* Change back to the normal cursor */
_setcursortype ( _NORMALCURSOR ) ;
}
```

13.

How do I write code that would get error number and display error message if any standard error occurs?

Ans: Following code demonstrates this.

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

main( )
{
char *errmsg ;
FILE *fp ;
fp = fopen ( "C:\file.txt", "r" ) ;
if ( fp == NULL )
{
errmsg = strerror ( errno ) ;
printf ( "\n%s", errmsg ) ;
}
}
```

Here, we are trying to open 'file.txt' file. However, if the file does not exist, then it would cause an error. As a result, a value (in this case 2) related to the error generated would get set in errno. errno is an external int variable declared in 'stdlib.h' and also in 'errno.h'. Next, we have called sterror( ) function which takes an error number and returns a pointer to standard error message related to the given error number.

14.

How do I write code to get the current drive as well as set the current drive?

Ans: The function getdisk( ) returns the drive number of current drive. The drive number 0 indicates 'A' as the current drive, 1 as 'B' and so on. The Setdisk( ) function sets the current drive. This function takes one argument which is an integer indicating the drive to be set. Following program demonstrates use of both the functions.

```
#include <dir.h>

main( )
{
int dno, maxdr ;
```

```
dno = getdisk( ) ;
printf ( "\nThe current drive is: %c\n", 65 + dno
) ;

maxdr = setdisk ( 3 ) ;
dno = getdisk( ) ;
printf ( "\nNow the current drive is: %c\n", 65 +
dno ) ;
}
```

15.

The functions memcmp( ) and memicmp( )

The functions memcmp( ) and memicmp( ) compares first n bytes of given two blocks of memory or strings. However, memcmp( ) performs comparison as unsigned chars whereas memicmp( ) performs comparison as chars but ignores case (i.e. upper or lower case). Both the functions return an integer value where 0 indicates that two memory buffers compared are identical. If the value returned is greater than 0 then it indicates that the first buffer is bigger than the second one. The value less than 0 indicate that the first buffer is less than the second buffer. The following code snippet demonstrates use of both

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

main( )
{
char str1[] = "This string contains some
characters" ;
char str2[] = "this string contains" ;
int result ;

result = memcmp ( str1, str2, strlen ( str2 ) ) ;
printf ( "\nResult after comapring buffer using
memcmp( )" ) ;
show ( result ) ;

result = memicmp ( str1, str2, strlen ( str2 ) ) ;
printf ( "\nResult after comapring buffer using
memicmp( )" ) ;
show ( result ) ;
}
```

```
show ( int r )
{
if ( r == 0 )
printf ( "\nThe buffer str1 and str2 hold
identical data" ) ;
if ( r > 0 )
printf ( "\nThe buffer str1 is bigger than buffer
str2" ) ;
if ( r < 0 )
printf ( "\nThe buffer str1 is less than buffer
str2" ) ;
}
```
