1.
Sample Code

```
#include
int main(void) {
{int i=10; goto lbl;}
{int i=20; {int i=30; lbl:;}
printf("%i", ++i);}
}
```

Refer to the **above sample code**. What does this program print?

1 10
2 11
3 21
4 31
5 An undefinedvalue

2.

```
short testarray[4][3] = { {1}, {2, 3}, {4, 5, 6} };
printf( "%d\n", sizeof( testarray ) );
```

Assuming a short is two bytes long, what will be printed by the **above code**?

1 It will not compile because not enough initializers are given.
2 6
3 7
4 12
5 24

3.

Which of the **following** choices most accurately describes variable DEFINITION (as opposed to declaration)?

1 The assignment of properties and storage space to a variable
2 The identification of a variable that resides elsewhere in the program
3 The assignment of storage space to a variable
4 The assignment of properties to a variable
5 The assignment of storage space to a variable whose properties have been specified external to the current file scope

4.

How do you include a system header file called

sysheader.h in a C source file?

1 #includefile
2 #include sysheader.h
3 #incl "sysheader.h"
4 #include
5 #incl

5.
Sample Code
f = fopen(fileName, "r");

From the **sample** **above**, which **one** of the **following** statements will set the file position for the file f to cause the next byte read from the file to be the last byte?

1 f = feof(f) - 1;
2 fsetpos(f, EOF - 1);
3 fseek(f, -1, SEEK_SET);
4 fseek(f, -1, EOF);
5 fseek(f, -1, SEEK_END);

6.
/* Initialize an **array** with zero values. */
void bzero (void * **b**, size_t len) {
char *buf = **b**;
int i;

for (i = 0; i <= len; i++) *(buf + i) = 0; } The function bzero(), defined **above**, contains a common programming error. Which **one** of the **following** correctly describes it? 1 The loop writes beyond the end of buf. 2 buf refers to an **array** and should be indexed as **one**; the pointer notation applied by the loop causes the wrong address to be dereferenced. 3 It is not permitted to declare objects of type size_t; this type is reserved for exclusive usage by the standard library's internal routines. 4 A generic pointer cannot be cast to a pointer of a more specific type. 5 The comparison between int i and size_t len is not permitted by Standard C.

7.
Sample Code
long factorial (long x)
{
return x * factorial(x - 1);
}

What will the function **above** return if called with x equal to 5?

1 0
2 15

3 120
4 The program will not compile.
5 The function will never return.

8.
Which one of the following describes a C character string?

1 A sequence of char objects in EBCDIC format
2 A sequence of printable ASCII characters
3 Any pointer to type char
4 An array of bytes terminated by NUL
5 A sequence of char objects in ASCII format

9.
#include %lt;stdlib.h>
int f(TYPE x) {
/*Allocates an array of two integers; returns 0
if the allocation succeeded, 1 otherwise.
Returns the allocated array in x.
*/
int *ret = (int*)malloc(sizeof(int[2]));
return ret ? *x=ret, 0 : 1;
}
Referring to the sample code above, how should TYPE be declared?

1 typedef int**TYPE;
2 typedef int*TYPE;
3 typedef int[]TYPE;
4 typedef &(int*)TYPE;
5 typedef int[]*TYPE;

10.
Which one of the following is a valid variable name?

1 Account_Number
2 Account Number
3 "Account Number"
4 Account^Number
5 Account-Number

11.
/* fp points to an open stream.
* Process a full stream of binary data. */
unsigned long process_binary_data(FILE *fp) {
char ch;

```
unsigned long count = 0UL;

assert(fp != NULL);
while ((ch = fgetc(fp)) != EOF) {
process_byte(ch);
count++;
}

return count;
}
```

The function process_binary_data(), defined **above**, may fail to process the entire stream under exceptional conditions. Which **one** of the **following** explains this error?

1 The disparity between the type of ch and the type of EOF causes the loop to execute forever.
2 The loop may fail to process the entire stream underlying fp because it does not check for errors.
3 It is incorrect to use the assignment operator in a loop condition. The programmer probably meant to use the equality operator.
4 The function body contains incorrectly nested parentheses.
5 count fails to contain the number of bytes actually processed.

12.

```
int x = 3;

if( x == 2 ); x = 0;
if( x == 3 ) x++;
else x += 2;
```

What value will x contain when the **sample** **code** **above** is executed?

1 1
2 2
3 3
4 4
5 5

13.

```
int x[] = {1, 2, 3, 4, 5};
int *ptr, **ptr2;

ptr = x;
ptr2 = &ptr;
```

Referring to the **sample** **code** **above**, how do you update x[2] to 10 using ptr2?

1 (**ptr2 + 2) = 10;
2 **(ptr2 + 2) = 10;
3 *(*ptr2 + 2) = 10;
4 *(&ptr2 + 2) = 10;

5 **(*ptr2 + 2) = 10;

14.
Which one of the following declarations ensures that ptr CANNOT be used to modify the string to which it points (although ptr itself may be changed to point to another string)?
1 static char *ptr = "Hello";
2 const char *ptr = "Hello";
3 char *ptr = const "Hello";
4 char * static ptr = "Hello";
5 char * const ptr = "Hello";

15.
int a [25];
int * p = a + 3;

Which one of the following is functionally equivalent to the code fragment above?

1 int a [25];
int * p = &a[3];

2 int a [25];
int * p = &(a + 3);

3 int a [25];
int * p = &a[2];

4 int a [25];
int * p = (&a)[3];

5 int a [25];
int * p = a[3];

16.
Which one of the following code fragments causes k to contain the value 16 at some point?

1 int j = 14;
int k = 1;
k += j+;

2 int k;
**((int **) &k) = 16;

3 int j;
int k = 4;
j = k <<>> 11;

k += j;

4 int j, k;
j = 2 << k =" j" j =" 256;" k =" 14;" k =" j">

17.
Sample Code

#define X(t,m) (size_t)((char*)&((t*)0)->m-(char*)(t*)0)

Consider the poorly named macro X(), defined **above**. Of which **one** of the **following** Standard C macros could X() be an implementation?
1 va_start()
2 putchar()
3 va_arg()
4 offsetof()
5 feof_unlocked()

18.
For which **one** of the **following** reasons did the writers of MS-DOS C compilers augment C by adding the reserved words __near, __far, and __huge?

1 They are reserved for usage by compiler writers and have no relevance to other programmers, hence the leading pair of underscores.
2 They indicate that the compiler should **place** objects in the data segment, on the stack, and on the heap, respectively.
3 They are specific to x86 architecture, and allow the programmer to override the default memory model for specific objects.
4 They allow the compiler to perform additional **code** optimizations on pointer values.
5 They indicate that an object should be placed near the lowest address, near the highest address, and that an object requires multiple pages, respectively.

19.
Sample Code
extern void *ptr1;
extern void *ptr2;

int compare(int n)
{
return ????;
}

What should replace the ???? in the **code** **above** to compare the first n bytes of the memory pointed to by ptr1 and ptr2 and return a non-zero value if they are equal?

1 memcmp(ptr1, ptr2, n)
2 !memcmp(ptr1, ptr2, n)

3 !strncmp(ptr1, ptr2, n)
4 memncmp(ptr1, ptr2, n)
5 strncmp(ptr1, ptr2, n)

20.
Sample Code
void memset (void * buf, int c, size_t len) ??< unsigned char * **b**; register int i; assert(buf);
debug("assert(buf) => TRUE??/n");
b = buf;
for (i = 0; i <>
??>

Consider the function memset(), defined **above**. Which **one** of the **following** is a completely true statement about the **code** shown **above**?

1 Some accesses performed on **b** are out-of-bounds.
2 Providing that debug() is declared and correctly used, the **code** compiles correctly under a Standard C compiler.
3 The three-character sequences starting with ?? were **inserted** by Microsoft Word.
4 The source **code** file containing memset() has probably become corrupted. This **code** clearly does not compile.
5 Assuming that debug() prints its string argument, the strings will run together on the same line.

21.

Which **one** of the **following** is the result type of the sizeof operator?

1 unsigned char
2 int
3 char
4 unsigned int
5 size_t

22.
Which **one** of the **following** best describes the Standard C convention regarding variables with names that start with an underscore (_)?
1 They are reserved for usage by standards committees, system implementers, and compiler engineers.
2 They are deprecated by Standard C and are permitted only for backward compatibility with older C libraries.
3 Applications programmers are encouraged to employ them in their own **code** in order to mark certain symbols for internal usage.
4 They are case-insensitive.
5 They are generally treated differently by preprocessors and compilers from other identifiers.

23.
Sample Code
int i,j;

```
int ctr = 0;
int myArray[2][3];

for (i=0; i<; i++) for (j=0; j<2;>
```

24.
Sample Code

```
int i = 4;

switch (i)
{
default:
;
case 3:
i += 5;

if ( i == 8)
{
i++;
if (i == 9) break;

i *= 2;
}

i -= 4;

break;

case 8:
i += 5;
break;
}

printf("i = %d\n", i);
```

What will the output of the **sample code above** be?

1 i = 5
2 i = 8
3 i = 9
4 i = 10
5 i = 18

25.
Which **one** of the **following** functions returns the string representation from a pointer to a time_t value?
1 localtime
2 gmtime
3 ctime
4 strtime
5 asctime

26.
Which one of the following is a true statement about an lvalue?

1 An lvalue is the result of an arithmetic operation involving quantities of type long int.
2 All lvalues can be used on the right side of an assignment statement.
3 An lvalue is, by definition, the value appearing on the rightmost side of an assignment statement.
4 By definition, an lvalue is the storage space indirectly referenced by a pointer.
5 An lvalue is any quantity capable of appearing on the left side of a shift operator.

27.
Which one of the following is a true statement about non-generic (void *) pointers?

1 For efficiency, pointer values are always stored in machine registers.
2 A pointer to one type may not be cast to a pointer to any other type.
3 Similarly typed pointers may be subtracted from each other.
4 They are always 32-bit values.
5 Similarly typed pointers may be added to each other.

28.
Which one of the following will read a character from the keyboard and will store it in the variable c?

1 c = getchar();
2 c = getchar(stdin);
3 getchar(&c)
4 getc(&c);
5 c = getc();

29.
Which one of the following is NOT a valid statement?

1 ++d+++e++;
2 f*=g+=h=5;
3 l-->>m<<--n; 4 int a(int a),b=0,c=a((c=b,++b)); 5 i->j<-k;

30.
```
int factorial (int x) {
extern jmp_buf jb;
int fact, chk;
if (!x) return 1;
fact = x * (chk = factorial(x - 1));
if (chk > fact) longjmp(jb, -1);
return fact;
}
```

```
int check_for_overflow (int x) {
extern jmp_buf jb;
if (setjmp(jb)) {
printf("discovered overflow in factorial(%d)\n", x);
return 0;
}
if (x < x =" 0;">
return factorial(x);
}
```

There is an error in check_for_overflow that may occasionally result in unexpected values in var jb. Which **one** of the **following** describes this error?

1 longjmp() cannot safely be used to escape from a recursive call chain.
2 The argument to check_for_overflow() should be qualified with volatile to ensure correct error reporting.
3 setjmp() and longjmp() **must** be invoked within the same stack frame. The result of longjmp() in this case is undefined.
4 The calls to setjmp() and longjmp() operate on different jump buffers, and therefore may have an undefined effect.
5 The factorial of zero (0) is incorrectly handled by factorial().

31.
The mostly redundant reserved word auto signifies that the compiler should allocate storage for a variable _____.

Which **one** of the **following** correctly completes the statement **above**?

1 Only on the stack
2 Only in a register
3 Only until the end of the block
4 Only in the data segment
5 Only on the heap

32.
Which of the **following** correctly enumerate predefined streams?

1 stddev, stdprn, and stdmon
2 stdin, stdout, and stderr
3 stdin, stdout, and stdterm
4 stdio and stderr
5 stdin, stdlib, and stderr

33.
Which **one** of the **following** Standard C functions can be used to sort a string **array**?
1 qsort
2 sort
3 quicksort
4 asort
5 There is no Standard C function for such a sort.

34.
An expression consisting solely of literals can appear anywhere that literals of the same type can appear.
Which one of the following correctly assesses the statement above?
1 The statement is true.
2 The statement is false. A constant expression may contain side effects that prevent it from being evaluated at compile time. Since it cannot be universally permitted, it is universally prohibited.
3 The statement is false. While a constant expression could logically appear wherever a literal could appear, the Standard C committee chose to allow constant expressions only in runtime contexts.
4 The statement is false. A constant expression cannot appear as the operand of sizeof or in the size definition of an array.
5 The statement is false. To simplify the design of the compiler, C compiler implementers are not obliged to evaluate constant expressions at compile time.

35.
```
/* sys/cdef.h */
#if defined(__STDC__) || defined(__cplusplus)
#define __P(protos) protos
#else
#define __P(protos) ()
#endif

/* stdio.h */
#include
div_t div __P((int, int));
```

The code above comes from header files for the FreeBSD implementation of the C library. What is the primary purpose of the __P() macro?

1 The __P() macro provides backward compatibility for K&R C compilers, which do not recognize Standard C prototypes.
2 The __P() macro has no function, and merely obfuscates library function declarations. It should be removed from further releases of the C library.
3 The __P() macro provides forward compatibility for C++ compilers, which do not recognize Standard C prototypes.
4 Identifiers that begin with two underscores are reserved for C library implementations. It is impossible to determine the purpose of the macro from the context given.
5 The __P() macro serves primarily to differentiate library functions from application-specific functions.

36.
Sample Code
```
int aFunction(int y)
{
int x = 3 * y;
```

```
return x;
}
```
In the sample code above, with what scope is the variable x implicitly declared?

1 extern
2 static
3 auto
4 volatile
5 register

37.
Which one of the following statements regarding macros is INCORRECT?
1 Macro definitions can contain expressions.
2 Macro definitions cannot be altered at runtime.
3 Macros can have parameters.
4 Macros are evaluated in the preprocessor.
5 Macros are evaluated at runtime.

38.
Sample Code

```
void listFile( FILE *f )
{
int c;
while( c = fgetc( f ) != EOF )
{
printf( "%d", c );
}
printf( "\n" );
}
```

What will be printed when the function above is called with a pointer to an open file that contains the three characters abc?

1 111
2 000
3 656667
4 abc
5 The characters ab followed by an infinite number of c's

39.
Which one of the following will turn off buffering for stdout?

1 setbuf(stdout, NULL);
2 setbuf(stdout, FALSE);
3 setvbuf(stdout, NULL);
4 setbuf(stdout, _IONBF);
5 setvbuf(stdout, _IONBF);

40.
Sample Code

```
char buf[ 50 ] = "Hello World";
char *ptr = buf + 5;
```

From the sample above, what is the proper way to copy 20 bytes from the location pointed to by ptr to the beginning of buf?

1 memcpy(buf, ptr, 20);
2 It cannot be done because the source and destination overlap.
3 strncpy(buf, ptr, 20);
4 That may cause an illegal memory access because it will read memory past the end of the string.
5 memmove(buf, ptr, 20);

41.
Which one of the following will declare a pointer to an integer at address 0x200 in memory?

1 int *x = &0x200;
2 int *x = *0x200;
3 int *x(&0x200);
4 int *x;
*x = 0x200;
5 int *x = 0x200;

42.

```
int *x;
x = (int *) 15;
```

Is the above code legal?

1 Yes; upon initialization, the number 15 is stored in a special pointer memory address space.
2 Yes; a new memory space is allocated to hold the number 15.
3 No; this syntax is not allowed.
4 Yes; the pointer x points at the integer in memory location 15.
5 No; this assigns the number 15 to an unallocated space in memory.

43.

```
x[2] = 5 vs.
2[x] = 5
```

Are x[2] and 2[x] identical in the sample code above? Why or why not?

1 Yes; both are identical because they are resolved to pointer references.
2 Yes; both are identical because the compiler will identify the x variable and make adjustments.
3 No; x[2] is correct, but 2[x] is invalid syntax.
4 No; both variable assignments have invalid syntax.
5 No; 2[x] is correct, but x[2] is invalid syntax.

44.
Sample Code
```
int x=1;
int y=x+++ ++x;
```

Refer to the **above** **sample** **code**. What is the value of y?

1 2
2 3
3 4
4 5
5 The value is implementation dependent because it is not defined in Standard C.

45.
Which **one** of the **following** functions is the Standard C functional equivalent of the AT&T Unix function rindex(), which returns a pointer to the last occurrence of a character in a string or NULL if no such character exists?

1 strrchr()
2 stridx()
3 strcspn()
4 strchr()
5 strridx()

46.
Which **one** of the **following** declarations ensures that ptr CANNOT be used to modify the string to which it points (although ptr itself may be changed to point to another string)?
1 const char *ptr = "Hello";
2 char *ptr = const "Hello";
3 char * static ptr = "Hello";
4 static char *ptr = "Hello";
5 char * const ptr = "Hello";

47.
Sample Code
```
char * strdup (const char * s) {
char * buf;
int len;

assert(s != NULL);

len = strlen(s);
buf = (char *) calloc(len + 1, sizeof(char));
memcpy(buf, s, len);

return buf;
}
```

The proposed implementation of strdup() **above** contains a runtime error that may NOT appear

consistently with each invocation. Which one of the following accurately describes this error?

1 The arguments to calloc() do not cause enough memory to be allocated for storing the contents of s.
2 If memory is scarce, calloc() may fail and return NULL. The code does not anticipate this condition.
3 memcpy() may corrupt data if used to copy ASCII strings.
4 buf is never NUL-terminated, and therefore cannot be used by C library functions affecting strings.
5 The function returns a pointer to dynamic memory. This practice should be avoided and always constitutes a memory leak.

48.
Sample Code
```
int a[5] = {1, 2, 3, 4, 5};
int *aPtr;
aPtr = a;
printf("element=%d\n", *(aPtr + 2));
```

What will be printed when the sample code above is executed?

1 element=1
2 element=2
3 element=3
4 element=4
5 element=5

9.
Sample Code
```
int my_func( int *a );

int main()
{
int b[3];
my_func( ???? );
return 0;
}
```

Referring to the sample code above, which one of the following must be inserted in place of the ???? to pass the array b to my_func?

1 (int *)b[0]
2 b
3 (int *)*b
4 (*b)[0]
5 *b

50.
Sample Code
```
void printTime( time_t *t )
{
????
}
```

Which one of the following can replace the ???? in the code above to print the time passed in t in human-readable form?

```
1 char s[ 100 ];
ctime( t, s );
printf( "%s\n", s );

2 printf( "%s\n", ctime( t ) );
3 printf( "%s\n", asctime( t ) );
4 printf( "%s", t );
5 char *s = ctime( t );
printf( "%s\n", s );
free( s );
```

51.
How do you declare a constant pointer to a constant string?

```
1 const* char const p;
2 const const char* p;
3 char const const* p;
4 const char const* p;
5 const char* const p;
```

52.
Sample Code #include

```
int ab=1;
int a=2;
int main(){
int b=3;
int ab=4;
printf("%i/*%i*/%i",a,b,ab);
}
```

Refer to the above sample code. What will be printed?

```
1 21
2 0/*4*/1
3 01
4 %i/*%i*/%i041
5 2/*3*/4
```

53.
Sample Code

```
int z;
int x = 5;
int y = -10;
int a = 4;
```

int b = 2;
z = x++ + ++y * b / a;

What number will z in the sample code above contain?

1 -3
2 -2
3 0
4 1
5 2

54.
Which one of the following functions allows an existing stream to be redirected?

1 setvbuf()
2 dup()
3 fopen()
4 popen()
5 freopen()

55.
Sample Code
int x = 0;

for (; ;)
{

if (x++ == 4) break;
continue;
}

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

What will be printed when the sample code above is executed?

1 x=0
2 x=1
3 x=4
4 x=5
5 x=6

56.
Sample Code
/* Eliminate all whitespace from the end of s. */
void rtrim (char * s) {
char * start, ws, text;

start = s, ws = text = s - 1;

```
while (*s != '\0') {
if (isspace(*s)) {
ws = s;
while (isspace(*s)) s++;
} else {
text = s;
while (*s != '\0' && !isspace(*s)) s++;
}
}

if (ws > text) *ws = '\0';
}
```

The function rtrim(), defined **above**, contains an implementation error. Which **one** of the **following** describes it?

1 The comma operator has a higher precedence than assignment; this produces an unintended effect.
2 rtrim() does not correctly handle the case where no character of s is whitespace.
3 rtrim() does not correctly handle the case where every character of s is whitespace.
4 ws and text are not assignment-compatible with s or NULL.
5 The loop and its subordinate statements do not adequately detect an end-of-string condition on s in all cases.

57.
Sample Code
```
struct customer *ptr = malloc( sizeof( struct customer ) );
```
Given the **sample** allocation for the pointer "ptr" found **above**, which **one** of the **following** statements is used to reallocate ptr to be an **array** of 10 elements?
1 ptr = realloc(ptr, 10 * sizeof(struct customer));
2 realloc(ptr, 10 * sizeof(struct customer));
3 realloc(ptr, 9 * sizeof(struct customer));
4 ptr += malloc(9 * sizeof(struct customer));
5 ptr = realloc(ptr, 9 * sizeof(struct customer));

58.
Sample Code
```
struct node {
int id;
int length;
struct node * next;
struct node * prev;
unsigned char data [1];
}
```

Consider struct node, an aggregate type defined **above**. Which **one** of the **following** might explain the declaration of its peculiar member data?

1 There is no difference between character unsigned char data and **array** unsigned char data [1], since each allocates only a single byte. Identical operations can be performed on both quantities. The choice was **one** of preference.
2 The programmer is declaring a bit field called data, which consists of only a single bit. struct node probably represents some hardware device.
3 data is probably used in conjunction with length and malloc() to create objects of variable size.

struct node is essentially a header for an object of indeterminate size.
4 The information provided by the definition of struct node is insufficient to formulate a guess about the purpose of the member data or its strange declaration.
5 Clearly the programmer has made a typo. If the programmer had intended to allocate only a single byte, he or she would have declared data as unsigned char data instead.

59.
Sample Code
char ca[]="abc\012\0x34";

Refer to the **above sample code**. What does strlen(ca) return, using a Standard C compiler?

1 3
2 4
3 5
4 10
5 12

60.
sizeof(L"Hello world!") == x

Consider the **code** fragment **above**. For which value of x is the expression **above** true?

1 sizeof(char *)
2 sizeof(wchar_t [])
3 12
4 13
5 13 * sizeof(wchar_t);

61.
unsigned i, *ip = &i;
*ip = *ip & ~*ip;

What does the **above sample code** do?

1 It sets ip to 0.
2 It sets all the bits of i to 0.
3 It sets i to the value of UINT_MAX.
4 It sets ip to NULL.
5 It produces an undefined result.

62.
Sample Code
char*(*(*x)(void))[];

What does the **above** statement declare?

1 An **array** of pointers to a function with no parameters returning a pointer to a string pointer
2 An **array** of pointers to a pointer to a function with no parameters returning a string
3 A pointer to an **array** of functions with no parameters returning a string
4 A pointer to a function with no parameters that returns a pointer to an **array** of strings
5 It is a syntactically incorrect statement.

63.

```
#include
#include
void f(int x) {
char b[] = "1234567";
strncpy(b, "abc", x);
{
int i=0;
for(; i
```

64.

Sample Code

```
char c = '\101';
```

What will the variable c contain in the **code above**?

1 It will not compile.
2 The four characters '\', '1', '0', '1'
3 A backslash '\'.
4 65 (which is the letter A in ASCII)
5 101 (which is the letter e in ASCII)

65.

Sample Code

```
void myFunc (int x)
{
if (x > 0) myFunc(--x);
printf("%d, ", x);
}

int main()
{
myFunc(5);
return 0;
}
```

What will the **above sample code** produce when executed?

1 5, 4, 3, 2, 1, 0,
2 0, 0, 1, 2, 3, 4,
3 4, 3, 2, 1, 0, 0,

4 1, 2, 3, 4, 5, 5,
5 0, 1, 2, 3, 4, 5,

66.
Sample Code
```
char s1[100];
char s2[100];

gets( s1 );
fgets( s2, sizeof(s2), stdin);
printf( "%d\n", strlen( s1 ) - strlen( s2 ) );
```

What will be printed when the **above** **code** is executed and the string "abcd" is entered twice on stdin?

1 -1
2 0
3 1
4 2
5 4

67.
Which **one** of the **following** correctly declares a pointer to an **array** of 12 characters?
1 char (* a) [12];
2 char (* a) [11];
3 char * a [11];
4 char * (a [12]);
5 char * (a [11]);

68.
Sample Code
```
void zero_array (char a [20]) {
size_t size;

assert(a);

size = sizeof(a);
memset(a, 0, size);
}
```

The function zero_array(), defined **above**, contains an error. Which **one** of the **following** describes it?

1 The result of sizeof(a) may be subject to a type cast that causes size to acquire an erroneous value. The call to memset() will probably clear the wrong number of bytes.
2 The assert() macro may incorrectly cause the **code** to abort if the host machine uses a constant other than zero (0) to represent the null pointer natively.
3 Standard C does not permit programmers to declare the size of the leftmost dimension of an **array** parameter. The compiler should print an error when parsing zero_array().
4 sizeof(a) evaluates to sizeof(char *) rather than twenty (20). The call to memset() will probably

clear the wrong number of bytes.
5 The second and third arguments to memset() are probably reversed.

69.
Sample Code
time_t currentTime = time(NULL);
printf("%s\n", ????);

Which **one** of the following can replace ???? in order for the above sample code to print out the current time as a human-readable string?

1 ctime(¤tTime)
2 asctime(&currentTime)
3 asctime(currentTime)
4 timestr(currentTime)
5 strtime(currentTime)

70.
Sample Code
#include
static double (*funcs[])(double) =
{
sin, cos, tan, asin, acos, atan, sinh, cosh, tanh
};

double computeTrigFunction(int index, double argument)
{
return ????;
}

Referring to the sample code above that should compute the value of a trigonometric function based on its index, what would be a replacement for the ???? to make the correct call?

1 *funcs[index](argument)
2 (*funcs)[index](argument)
3 funcs(argument)[index]
4 *(funcs[index](argument))
5 funcs[index](argument)

71.
Sample Code
file inc.h contains only the following line:
struct x {int d; char c;}

file main.c contains only the three following lines:

#include
#include "inc.h"
main(int n, char**a) { exit(0); }

Refer to the above sample code. The program is supposed to be Standard C. What is the problem with main in this code?

1 main does not return a value.
2 The parameters of main must be named argc and argv.
3 main is missing a return type.
4 The parameters of main are not used.
5 The type of the second parameter of main is invalid.

72.
Sample Code
```
int x = 5;
int y = 2;
char op = '*';
switch (op)
{
default : x += 1;
case '+' : x += y;
case '-' : x -= y;
}
```

After the sample code above has been executed, what value will the variable x contain?

1 4
2 5
3 6
4 7
5 8

1.
```
class A {
};

class B {
protected:
friend class A;
};

class C {
public:
friend class B;
};
```

Referring to the sample code above, assuming the above classes had data members, what names of C's members could be used in declarations of members of A?
1) Only private members
2) Only protected members
3) All of C's data members
4) Only public members
5) None of C's data members

courtesy:***DEVFYI - Developer Resource - FYI***