

1.

```
int z,x=5,y=-10,a=4,b=2;
```

```
z = x++ - --y * b / a;
```

What number will z in the sample code above contain?

1. 5 2. 6 3. 10 4. 11 5. 12

2. With every use of a memory allocation function, what function should be used to release allocated memory which is no longer needed?

1. unalloc() 2. dropmem() 3. dealloc() 4. release() 5. free()

3.

```
void *ptr;
```

```
myStruct myArray[10];
```

```
ptr = myArray;
```

Which of the following is the correct way to increment the variable "ptr"?

1. ptr = ptr + sizeof(myStruct); 2. ++(int*)ptr;
3. ptr = ptr + sizeof(myArray); 4. increment(ptr);
5. ptr = ptr + sizeof(ptr);

4.

```
char* myFunc (char *ptr)
```

```
{
```

```
ptr += 3;
```

```
return (ptr);
```

```
}
```

```
int main()
```

```
{
```

```
char *x, *y;
```

```
x = "HELLO";
```

```
y = myFunc (x);
```

```
printf ("y = %s \n", y);
```

```
return 0;
```

```
}
```

What will print when the sample code above is executed?

1. y = HELLO 2. y = ELLO 3. y = LLO
4. y = LO 5. x = O

5.

```
struct node *nPtr, *sPtr; /* pointers for a linked list. */
```

```
for (nPtr=sPtr; nPtr; nPtr=nPtr->next)
```

```
{
```

```
free(nPtr);
```

```
}
```

The sample code above releases memory from a linked list. Which of the choices below accurately describes how it will work?

1. It will work correctly since the for loop covers the entire list.
2. It may fail since each node "nPtr" is freed before its next address can be accessed.
3. In the for loop, the assignment "nPtr=nPtr->next" should be changed to "nPtr=nPtr.next".
4. This is invalid syntax for freeing memory.
5. The loop will never end.

6. What function will read a specified number of elements from a file?

1. fread()
2. getline()
3. readfile()
4. **fread()**
5. gets()

7.

"My salary was increased by 15%!"

Select the statement which will EXACTLY reproduce the line of text above.

1. printf("\nMy salary was increased by 15/%!\n");
2. printf("My salary was increased by 15%!\n");
3. printf("My salary was increased by 15'%'!\n");
4. **printf("\nMy salary was increased by 15%%!\n");**
5. printf("\nMy salary was increased by 15'%'!\n");

8. What is a difference between a declaration and a definition of a variable?

1. Both can occur multiple times, but a declaration must occur first.
2. There is no difference between them.
3. A definition occurs once, but a declaration may occur many times.
4. **A declaration occurs once, but a definition may occur many times.**
5. Both can occur multiple times, but a definition must occur first.

9.

```
int testarray[3][2][2] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
```

What value does testarray[2][1][0] in the sample code above contain?

1. 3
2. 5
3. 7
4. 9
5. **11**

10.

```
int a=10,b;
```

```
b=a++ + ++a;
```

```
printf("%d,%d,%d,%d",b,a++,a,++a);
```

what will be the output when following code is executed

1. 12,10,11,13
2. 22,10,11,13
3. 22,11,11,11
4. 12,11,11,11
5. **22,13,13,13**

11. What number of digits that can be accurately stored in a float (based on the IEEE Standard 754)?

- a) **6**
- b) 38
- c) An unlimited number

12. Which of the following is evaluated first:

- a) &&
- b) ||
- c) **!**

13. What does 7/9*9 equal (in C and C++)?

- a) 1
- b) 0.08642
- c) **0**

14. Which is not valid in C?

- a) `class aClass{public:int x;};b) /* A comment */` c) `char x=12;`

15. Which of the following is not a valid declaration for main()?

- a) `int main()` b) `int main(int argc, char *argv[])`
c) **They both work**

16. Evaluate as true or false: `!(1 &&0 || !1)`

- a) **True** b) False c) Invalid statement

17. Which command properly allocates memory?

- a) `char *a=new char[20];` b) `char a=new char[20];`
c) `char a=new char(20.0);`

18. What operator is used to access a struct through a pointer?

- a) `->` b) `>>` c) `*`

19. Which is not an ANSI C++ function?

- a) `sin()` b) `kbhit()` c) `tmpnam()`

20. True or false, if you keep incrementing a variable, it will become negative?

- a) True b) False c) It depends on the variable type

21. What character terminates all character array strings

- a) `\0` b) `.` c) `\END`

22. If you push the numbers (in order) 1, 3, and 5 onto a stack, which pops out first?

- a) **5** b) 1 c) 3

23. What does `strcat(an_array, "This");` do?

- a) Copies "This" into an_array b) **Adds "This" to the end of an_array**
c) Compares an_array and "This"

24. Evaluate:

```
int fn(int v)
{
    if(v==1 || v==0)
        return 1;
    if(v%2==0)
        return fn(v/2)+2;
    else
        return fn(v-1)+3;
}
for fn(7);
```

a) 10 b) 11 c) 1

25. Evaluate the following: $22\%5$

a) 2 b) 4 c) 0

26. Which of the following data structures is on average the fastest for retrieving data?

a) Binary Tree b) Hash Table c) Stack

27. What is the output:

```
int v()
{
    int m=0;
    return m++;
}
int main()
{
    cout<<v();
}
```

a) 1 b) 0 c) Program is illegal

28. What function initializes variables in a class:

a) Constructor b) Destructor c) Constitutor

29. Which datatype can store decimal numbers?

a) unsigned int b) char c) float

30. What does `cout<<(0==0)` print out?

a) 0 b) 1 c) Compiler error: Lvalue required

31. What does `getch()` do according to the ANSI C++ standard?

a) Reads in a character b) Checks the keyboard buffer
c) Nothing in particular (`getch()` is not an ANSI C++ Function).

32. If all is successful, what should main return?

a) 0 b) 1 c) void

33. Which sort is best for the set: 1 2 3 5 4

a) Quick Sort b) Bubble Sort c) Merge Sort

34. What does the following code do: `int c=0; cout<<c++<<c;`

a) Undefined b) 01 c) 00

35. What is the maximum value of a unsigned char?

a) 255

b) 256

c) 128

36. In `int main(int argc, char *argv[])` what is `argv[0]` usually going to be?

a) The first argument passed into the program

b) The program name

c) You can't define main like that.

37. In which header file does one find `isalpha()`

a) `conio.h`

b) `stdio.h`

c) `ctype.h`

38. What will happen:

```
int x;  
while(x<100)  
{  
    cout<<x;  
    x++;  
}
```

a) Outputs 0123..99

b) Outputs 0123...100

c) The output is undefined.

38. Will a C compiler always compile C++ code?

a) Yes.

b) No.

c) Only optimized compilers.

39. Which is not a valid keyword:

a) `public`

b) `protected`

c) `guarded`

40. What is the correct syntax for inheritance

a) `class aclass : public superclass`

b) `class aclass inherit superclass`

c) `class aclass <- superclass`

41. What does the following do: `for(;;) ;`

a) Illegal

b) Loops forever

c) Ignored by compiler...not illegal

42. Of the numbers 12 23 9 28 which would be at the top of a properly implemented maxheap?

a) 28

b) 9

c) Any of them could be

43. What does the `break;` do in the following?

```
void afunction()  
{  
    if(1)  
    {  
        break;  
        a_function();  
        cout<<"Err";  
    }  
}
```

a) Breaks out of the if statement b) Exits the function c) **Nothing (Compiler error)**

44. What are pointers, when declared, initialized to?

a) NULL b) Newly c) **Nothing.**

45. What is the last index number in an array of 100 characters?

a) 100 b) **99** c) 101

46. Would you rather wait for quicksort, linear search, or bubble sort on a 200000 element array? (Or go to lunch...)

a) Quicksort b) **Linear Search** c) Bubble Sort

47. Which of the following data structures uses the least memory?

a) struct astruct

```
{  
int x;  
float y;  
int v;  
};
```

b) **union aunion**

```
{  
int x;  
float v;  
};
```

c) char array[10];

48. What does the following do:

```
void afunction(int *x)  
{  
x=new int;  
*x=12;  
}  
int main()  
{  
int v=10;  
afunction(&v);  
cout<<v;  
}
```

a) Outputs 12 b) **Outputs 10** c) Outputs the address of v

49. What do nonglobal variables default to:

a) **auto** b) register c) static

50. Which header file allows file i/o with streams?

a) iostream b) **fstream** c) fileio.h

51. What is the outcome of cout<<abs(-16.5);

a) 16

b) 17

c) 16.5

52. What will `strcmp("Astring", "Astring");` return?

a) A positive value

b) A negative value

c) Zero

53. Evaluate `!(1&&1||1&&0)`

a) Error

b) True

c) False

54. What header file is needed for `exit()`;

a) `stdlib.h`

b) `conio.h`

c) `dos.h`

55. What ANSI C++ function clears the screen?

a) `clrscr()`

b) `clear()`

c) Its not defined by the ANSI C++ standard

56. Will a recursive function without an end condition ever quit, in practice?

a) Compiler-Specific

b) No

c) Yes

57. How long does this loop run: `for(int x=0; x=3; x++)`

a) Never

b) Three times

c) Forever

58. When will this line fail to compile:

`new myObj[100];`

a) Never

b) When `myObj` is too large to fit into memory

c) When `myObj` has no default constructor

Question #2

Assuming that `myObj` is less than 1000 bytes, is there anything wrong with this code?

`char x[1000];`

`myObj *obj = reinterpret_cast(x);`

`new (obj) myObj;`

a) Nope, it works fine

b) Yes, there could be byte alignment issues

c) Yes, the syntax for calling `new` is incorrect

Question #3

What is the functional difference between

`myObj *x = new myObj[100];`

`delete x;`

and

`myObj *x = new myObj[100];`

`delete [] x;`

a) There is none; they both work as expected

b) They both do nothing.

c) The first will not invoke all `myObj` destructors

Question #4

What is wrong with the following code?

```
int * x = (int *) malloc(100 * sizeof(int));
```

```
x = realloc(x, sizeof(int) * 200);
```

- a) If realloc fails, then the original memory is lost
- b) Nothing, realloc is guaranteed to succeed (by returning the original pointer)
- c) Nothing, realloc frees the original memory passed to it

Question #5

What is one possible symptom of having called free on the same block of memory twice?

- a) Nothing, calling free twice on one block of memory always works fine
- b) Malloc might always return the same block of memory
- c) You cannot free any memory in the future

Question #6

What's wrong with this line of code?

```
delete NULL;
```

- a) It causes a segmentation fault when delete tries to access NULL
- b) Nothing
- c) It is undefined behavior

Question #7

Assuming this code were being compiled using a standards-conforming C++, what is wrong with it?

```
int * x = malloc(100 * sizeof(int));
```

```
x = realloc(x, sizeof(int) * 200);
```

- a) malloc is undefined in C++, you can only allocate memory using new
- b) Invalid conversion from a void* to an int*
- c) Nothing is wrong with this code

Question #8

What happens if the normal new operator fails?

- a) This is left up to the implementation to decide
- b) It returns NULL
- c) It throws an exception

Question #9

What is a difficulty that arises with handling out of memory errors?

- a) Many cleanup operations require extra memory
- b) Running out of memory rarely happens in systems that interact with users
- c) Once you've run out of memory, your program cannot continue

Question #10

What is the result of this code?

```
myObj *foo = operator new(sizeof(foo));
```

- a) That's not legal syntax!
- b) foo has memory allocated for it, but is not constructed
- c) foo has memory allocated for it, and is fully constructed

Question #1

Which of the following is illegal:

a)

```
template <class T> func(T x) {}
```

```
template <class T> func<T*>(T* x) {}
```


b)
template <class T>
class myObject {};
c)
template <class T>
class myObj { template <class R> memFunc() {} };

Question #2

What problems can a templated member function cause?

- a) They're just not legal
- b) They're hard to use
- c) **They allow violations of encapsulation**

Question #3

When must template functions have explicit template parameters?

- a) Always
- b) **When the template types cannot be inferred**
- c) Never, the template types can always be inferred

Question #4

Are templates conceptually related to polymorphism?

- a) Nope
- b) Only when the template types are objects
- c) **Yes, but compile-time polymorphism**

Question #5

In general, is it possible to completely hide the source code of a library written using templates?

- a) Yes, but using export feature
- b) **No, pretty much never**
- c) Yes, all the time

Question #6

When are templates usually instantiated?

- a) At runtime
- b) **At compile time**
- c) At link time

Question #7

Which of the following describes a potentially surprising result of using templates?

- a) Slower programs
- b) Poor variable naming in the debugger
- c) **Increased executable size in comparison to the code base**

Question #8

Which of the following is an invalid template declaration:

- a) template <int x> int func() {return x;}
b) **template <double x> double func() {return x;}
c) template <typename x> void func(x t) {}**

Question #9

What is the result of trying to run this program:

```
#include <iostream>
template <int T>
struct X
{
    enum val {v = T * X<T-1>::v };
};
```

```
template <>
struct X<0>
{
    enum val {v = 1 };
};
```

```
int main() { std::cout<<X<5>::v; }
```

- a) Compilation error
- b) Link error

c) 120

Question #10

Given the below code, what happens when a method invokes callFunc on an object of type obj?

```
template <class X> func(X val) {}
template <> func<double>(double val) {}
```

```
class obj
{
    public:
        callFunc() { func(4.5); }
    private:
        func(int val) {}
};
```

- a) func(int val)
- b) template <class X> func(X val)
- c) template <> func<double>(double val)