



**READ THIS NOW!**

- Print your name in the space provided below. For the 10:10 section code a group of '1' for the 12:20 section code a group of '2'.
- Print your name and ID number on the Opscan form; be sure to code your ID number on the Opscan form. Code **Form A** on the Opscan.
- Choose the single best answer for each question — some answers may be partially correct. If you mark more than one answer, it will be counted wrong.
- Unless a question involves determining whether given C++ code is syntactically correct, assume that it is valid. The given code has been compiled and tested, except where there are deliberate errors. Unless a question specifically deals with compiler `#include` directives, you should assume the necessary header files have been included.
- Be careful to distinguish integer values from floating point (real) values (containing a decimal point). In questions/answers which require a distinction between integer and real values, integers will be represented without a decimal point, whereas real values will have a decimal point, [ 1704 (integer), 1704.0 (real)].
- The answers you mark on the Opscan form will be considered your official answers.
- When you have completed the test, sign the pledge at the bottom of this page and turn in the test.
- This is a closed-book, closed-notes examination. No calculators or other electronic devices may be used during this examination. You may not discuss (in any form: written, verbal or electronic) the content of this examination with any student who has not taken it. You must return this test form when you complete the examination. Failure to adhere to any of these restrictions is an Honor Code violation.
- There are 25 questions, equally weighted. The maximum score on this test is 100 points.

**Do not start the test until instructed to do so!**

Print Name (Last, First) \_\_\_\_\_

**Solution**

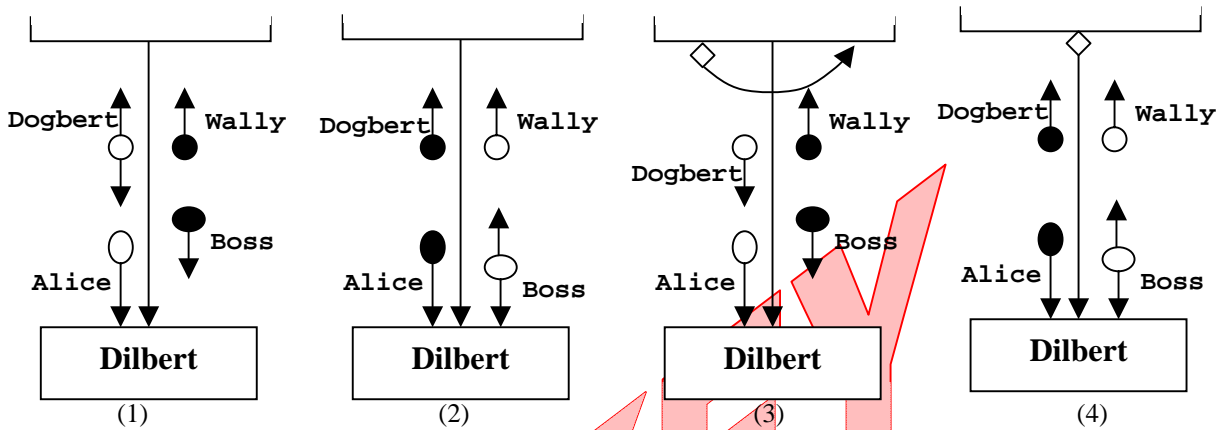
**Pledge:** On my honor, I have neither given nor received unauthorized aid on this examination.

**N. D. Barnette**

signature

**I. Design Representation**

Use the following partial Structure Chart diagrams and responses below as answers for the next 2 questions:



- (5) 1 & 2      (6) 2 & 3      (7) 3 & 4      (8) 1 & 3      (9) 2 & 4      (10) None of the above

Do not make any assumption about variables that are not shown on the chart. Given the following variable definitions:

```
int Dogbert, Wally,
    Alice, Boss;
```

#1 Which of the above structure chart diagrams for Dilbert() correctly models the code segment below?

```
Dilbert(Dogbert, Wally, Alice, Boss);
if (Wally)
    //code under control of if
```

```
void Dilbert(int& Dogbert, int& Wally,
            int Alice, int Boss) {
    if (Boss < 1)
        //code under control of if
```

**Analysis:**  
The reference parameters, Dogbert & Wally, could be either output or input/output parameters. The value parameters, Alice & Boss, can only be input parameters. Since Wally is used for control in the calling and Boss is used for control in Dilbert, they must be control parameters and the other parameters, (Dogbert and Alice) must be data parameters.  
  
Given those observations, only structure chart above that satisfies the analysis is (1).

#2 Which of the above structure chart diagrams for Dilbert() correctly models the code segment below?

```
if (Dogbert)
    Dilbert(Dogbert, Wally,
            Alice, Boss );
```

```
void Dilbert(int& Dogbert, int& Wally,
            int Alice, int& Boss) {
    if (Alice)
        //code under control of if
```

**Analysis:**  
The reference parameters, Dogbert, Wally & Boss could be either output or input/output parameters there is no way to tell from the partial code segment. The value parameter, Alice, can only be input parameters. Since Dogbert and Alice are used for control the other parameters must be data parameters.  
  
Given those observations, only structure chart above that satisfies the analysis is (4).

**II. Pointers**

Assume the following declarations:

```
const int CAPACITY = 100;
int r = 0, w[CAPACITY]={0};

int* p; int* q;
```

Use the responses:

- (1) Valid (2) Invalid

for the next 6 questions (#3 - #9). Considering each statement below **independently**, determine whether each statement would compile (not link) without errors after the statement:

*KEY*

```
p = new int[CAPACITY];

#3 delete [] w; (2) Invalid (Static Arrays cannot be deallocated.)
#4 q = &p[ -1 + CAPACITY]; (1) Valid (q = address of last element of p array)
#5 *p = *(&w[CAPACITY - 1]); (1) Valid (equivalent to p[0] = w[99];)
#6 p[1] = w*2+1; (2) Invalid (Pointer multiplication is not defined.)
#7 p[CAPACITY-1] = w[CAPACITY-1]; (1) Valid (equivalent to *(p+99) = *(w+99);)
#8 *(w+1) = *(p+1); (1) Valid (equivalent to w[1] = p[1];)
```

#9 Identify the most serious logical error that best identifies what occurs in the code fragment:

- (1) Alias pointer exists  
 (2) **Dangling pointer exists**  
 (3) Illegal memory address reference  
 (4) Memory garbage exists  
 (5) Undefined pointer dereferenced  
 (6) No logical error occurs

```
char *c = new char[4];
char *d = &c[0];
delete [] c;
d[0]= 'W'; d[1]= 'X';
d[2]= 'Y'; d[3]= '\0';
```

**D is an alias that is dangling after c is deleted.**

#10 Identify the most serious logical error that best identifies what occurs in the code fragment:

- (1) Alias pointer exists  
 (2) Dangling Reference exists  
 (3) Illegal memory address reference  
 (4) Memory garbage exists  
 (5) **Undefined pointer dereferenced**  
 (6) No logical error occurs

```
char *c = new char[2];
*c = 'A'; *(c+1)= '\0';
delete c;
*c = 'B';
```

**Only the [0] position would be deleted, but regardless after any delete the pointer is**

#11 What value is printed by the code fragment below?

```
const int SIZE = 5;
float* r; float * f;

r = new float [SIZE]; // assume allocation starts at address 00010000

for (int i = 0; i < SIZE; i++)
    r[i] = float(i);
f = r;
f = f + 1;
cout << " f = " << f << endl;
```

r == &r[0] which is 0001000,  
f = r; sets f to 0001000,  
f = f + 1 == (&r[0]) + 1, which is pointer  
addition. Since r is a float array and since  
each float takes 4 bytes of memory:  
f = f + 1 == 00001004

- (1) 00010000      (2) 00010001      **(3) 00010004**      (4) 00010005  
(5) 0.0            (6) 1.0            (7) 2.0            (8) None of the above

Consider the following code:

```
void DelMem (int arr[],
             int deinit, int dim); //Deallocate array memory & zero
void DelMem( int arr[],
             int deinit, int dim)
const int DIM = 5;
void main() {
int* a = new(nothrow) int[DIM]; //zero memory for safety
//use array
DelMem( a, 0, DIM);
}
for (int* i=arr; dim >0; i++, dim--)
    *i = deinit; //deinitialize
delete [] arr; //delete array
```

#12 In the code above, how is the array int pointer variable a being passed to the DelMem ( ) function?

- (1) by value                      (2) by reference                      (3) by const reference  
**(4) as a const pointer**      (5) as a pointer to a const target      (6) as a const pointer to a const target  
(7) none of the above

#13 Unfortunately the above call to DelMem ( ) does not function as intended. Select the statement below that best describes how to fix the problem.

- (1) the dim parameter must not be decremented and used for loop control termination, a temporary local variable should be defined and used for this purpose.  
(2) the integer array parameter, a, must be passed as a pointer to a const target parameter to allow the dynamic array memory to be deallocated by delete.  
(3) the integer array parameter, a, must be passed as a const pointer parameter to prevent the for loop in DelMem ( ) from accidentally resetting the array dimension when dim is decremented.  
**(4) the integer array parameter, a, must be passed as a reference pointer parameter to allow the changes made by DelMem ( ) to take effect and prevent a compilation error from occurring.**  
(5) none of the above

Use the responses:

- (1) Valid
- (2) Invalid

for the next 6 questions (#14 - #19). Considering each numbered question statement in the function below separately, determine whether each statement would be valid or invalid:

Assume the following function declarations:

```
void FN1(const int* const FormalParm);
void FN2(int* const FormalParm);
void FN3(const int* FormalParm);

void G(int*& IntArrayPtr) {
    int *a = IntArrayPtr;
    int *b = IntArrayPtr;
    int *c = IntArrayPtr;

    FN1(a);
    FN2(b);
    FN3(c);
} // Q U E S T I O N S
// -----

void FN1(const int* const FormalParm){
    int arr[6] = {0, -1, -2, -3, -4, -5};

    FormalParm = arr; // #14: (1)Valid or (2)Invalid ?
    FormalParm[0] = 6; // #15: (1)Valid or (2)Invalid ?
}

void FN2(int* const FormalParm) {
    int arr[6] = {0, -1, -2, -3, -4, -5};

    FormalParm = arr; // #16: (1)Valid or (2)Invalid ?
    FormalParm[0] = 6; // #17: (1)Valid or (2)Invalid ?
}

void FN3(const int* FormalParm) {
    int arr[6] = {0, -1, -2, -3, -4, -5};

    FormalParm = arr; // #18: (1)Valid or (2)Invalid ?
    FormalParm[0] = 6; // #19: (1)Valid or (2)Invalid ?
}
```



FormalParm is a constant pointer to a constant target neither it nor its target can be

FormalParm is a constant pointer to an int target thus its pointer value can NOT be re-assigned.

FormalParm is a pointer to a constant, its target can NOT be re-assigned.

**III. Class Basics**

Assume the following class declaration and implementation:

```

class FloorLamp {
private:
    bool light; //true: light is on
    int watts; //brightness: 0..100
public:
    FloorLamp();
    FloorLamp(bool button, int level);
    void on();
    void off();
    bool onoff();
    void dim(int delta);
    int brightness();
};

FloorLamp:: FloorLamp() {
    light = false; //light off
    watts = 50; //half intensity
}

FloorLamp:: FloorLamp(bool button,
                      int level) {
    light = button;
    watts = level;
}

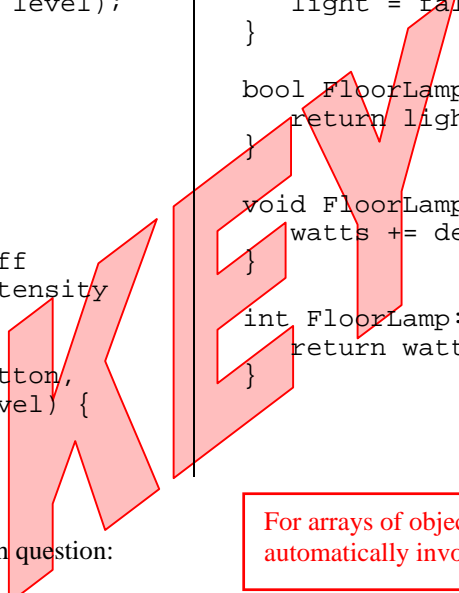
void FloorLamp::on() {
    light = true;
}

void FloorLamp::off() {
    light = false;
}

bool FloorLamp::onoff() {
    return light;
}

void FloorLamp:: dim(int delta) {
    watts += delta; //positive||negative
}

int FloorLamp:: brightness() {
    return watts;
}
    
```



For arrays of objects the default constructor is automatically invoked upon each object in the array.

Circle the number of the best answer to each question:

#20 How many function invocation(s), (i.e. function executions), does the following statement cause:

```
FloorLamp Lamps[10];
```

- (1) 1                      (2) 2                      (3) 9                      **(4) 10**
- (5) 11                    (6) 12                    (7) 20                    (8) Zero

#21 Given the object definitions at the right, which of the following object statements are valid?

```
FloorLamp Table(true, 25), Desk;
```

- (1) bool SameLampState = ( Desk == Table );                      (2) FloorLamp Room = Table;
- (3) Desk = Table;                      (4) Table->dim(-25);                      (5) cout << Desk;
- (6) 1 & 2                      **(7) 2 & 3**                      (8) 3 & 4
- (9) 4 & 5                      (10) All are valid

For objects C++ does not provide for equality comparison or output automatically. C++ does provide for object assignment and initialization, (shallow copying), of objects.

#22 Which of the member functions in the FloorLamp class should have been declared as const member functions?:

- (1) FloorLamp();
- (2) FloorLamp(bool button, int level);
- (3) void on();
- (4) void off();
- (5) bool onoff();
- (6) void dim(int delta);
- (7) int brightness();
- (8) 1 & 2
- (9) 3, 4 & 5
- (10) 5 & 7**

Only the onoff and brightness member functions do not change any of the private data.

#23 Given the object definition at the right, which of the following object statements would cut power to the lamp?

```
FloorLamp Banker(true, 75);
```

- (1) Banker.light = false;
- (2) Banker.watts = 0;
- (3) Banker.dim(-100);
- (4) Banker.off();**
- (5) None of the above

The off member function turns the light off.

#23 How many mutator member functions does the FloorLamp class declaration contain?

- (1) 1
- (2) 2
- (3) 3 //see #22 above for explanation.**
- (4) 4
- (5) 0
- (6) None of the above

#25 Given the function definition at the right, what do the following statements accomplish:

```
bool TooHot (FloorLamp lamp);  
  
void main () {  
    FloorLamp book(true, 100);  
    book.dim(25);  
    if (TooHot(book) )  
        cout << "***Book Lamp Setting Too Hot***";  
}
```

```
bool TooHot (FloorLamp lamp) {  
    return(lamp.watts > 100 );  
} // TooHot
```

TooHot() is NOT a class function member. It CANNOT access private class members.

- (1) causes main() to display a warning message.
- (2) causes a execution error when the dim() member function is invoked.
- (3) causes a compilation error.**
- (4) contains a logic error by setting the lamp watts above 100.
- (5) None of these