READ THIS NOW!

- Print your name in the space provided below.
- 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!

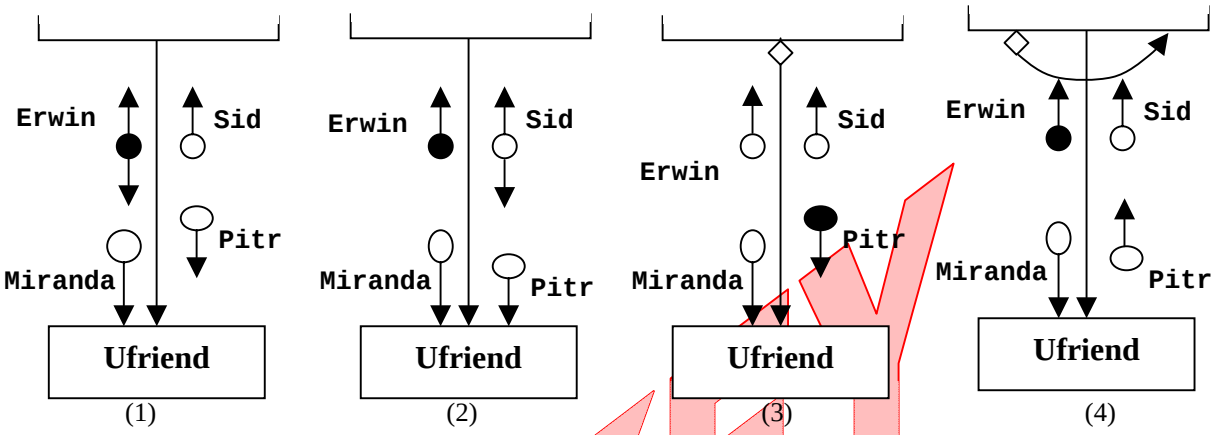
Print Name (Last, First) **Solution**

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 below as answers for the next 2 questions:



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

```
bool   Erwin,  Sid, Greg;
int    Pitr, Miranda;
```

#1 Which of the above structure chart diagrams for `Ufriend()` cor

```
if (Greg)
   Ufriend(Erwin, Sid, Miranda, Pitr);
```

```
void Ufriend(bool& Erwin, bool& Sid,
             int Miranda, int  Pitr) {
   while (Pitr > 0)
       //code under control of while
```

Analysis:

`Greg` is NOT passed so it has no representation on the structure chart. The reference parameters, Erwin & Sid, could be either output or input/output parameters there is no way to tell from the partial code segment. The value parameters, Miranda & Pitr, can only be input parameters. Since only Pitr is used for control the other parameters must be data parameters.

Given those observations, only structure chart above that satisfies the analysis is **(3)**.

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

```
Ufriend (Erwin, Sid, Miranda, Pitr);
if (Erwin)
       //code under control of if
```

```
void Ufriend(bool& Erwin, bool& Sid,
             int Miranda, int  Pitr) {
   while (Erwin)
       //code under control of while
```

Analysis:

The reference parameters, Erwin & Sid, could be either output or input/output parameters, however since Erwin is used for control in both functions it must be an inout/output control parameter. The value parameters, Miranda & Pitr, can only be input parameters. Since only Erwin is used for control the other parameters must be data parameters.

Given those observations, only structure chart above that satisfies the analysis is **(1)**.

II. Pointers

Assume the following declarations:

```
const int SIZE = 10;
int x = 0, y[SIZE]={0};

int* a; int* b;
```

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:

```
a = new int[SIZE];
```

#3
`delete [] y;` ***(2) Invalid** (Static Arrays cannot be deallocated.)*

#4
`a = NULL;` ***(1) Valid***

#5
`y++;` ***(2) Invalid** (Array pointers are constant)*

#6
`a++;` ***(1) Valid***

#7
`b = &a + SIZE;` ***(2) Invalid** (&a is a pointer to a pointer, i.e. int**, the resulting expression cannot be assigned to be which is just a pointer, i.e. int*)*

#8
`*(a + 1) = y[SIZE – SIZE];` ***(1) Valid***

#9 Identify the most serious type of logical error that occurs as a result of the statements:

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

```
int *p = new int[5];
int *q = p;
p--;
delete [] q;
cout << p << endl;
p = q = NULL;
```

#10 Identify the most serious type of logical error that occurs as a result of the statements:

(1) Alias pointer exists
(2) Dangling Reference exists
(3) Illegal memory address reference
(4) Memory garbage/leak exists
(5) Undefined pointer dereferenced
== 0.5 credit
(6) No logical error occurs

p is NULL when it is dereferenced.

```
int *p = new int[5];
int *q = p;
p--;
delete [] q;
p = q = NULL;
cout << *p << endl;
```

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

```
const int SIZE = 5;
int* x; int* y; int i;

x = new int[SIZE];  // assume allocation starts at address 00002000

for (i = 0; i < SIZE; i++)
      x[i] = i;
y = x + 1;
cout << " y = " << &y << endl;
```

(1) 1 (2) 00002001 **(3) 00002004**

(4) 4 (5) 00002008 (6) None of the above

x == &x[0] which is 00002000, y = x + 1 == (&x[0]) + 1, which is pointer addition. Since x is an int array and since each int takes 4 bytes of memory: y = x + 1 == 00002004

Consider the following code:

```
void GetMem (int* const arr,
             int size, int init);

const int SIZE = 10;
void main() {
int* a;

  GetMem( a, SIZE, -1);

  for (int i =0; i < SIZE; i++)
      cout << a[i] << " ";

  delete [SIZE] a;
}
```

```
//allocate array memory & initialize
void GetMem( int* const arr,
             int size, int init)
{
  arr = new int[size]; //get new array
  for (int* i=&(arr[0]); size>0; i++, size--)
      *i = init;       //initialize
  return;
}
```

#12 In the code above, how is the array int pointer variable `a` being passed to the `GetMem()` 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 `GetMem()` may not function as intended. Select the statement below that best describes how to fix the problem.

(1) the size 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 -1 parameter must not be passed as the init parameter to prevent it from being corrupted when the init parameter is decremented.

(3) the integer pointer parameter, a, must be passed as a pointer to a constant target to prevent the `for` loop in `GetMem()` from accidentally resetting the array dimension when size is decremented.

(4) the integer pointer parameter, a, must be passed as a reference pointer parameter to allow the changes made by `GetMem()` to be performed and prevent an illegal assignment occurring when the function is compiled.

(5) none of the above (`GetMem()` does function as intended)

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:

```
const int SIZE = 6;

void main() {

  double a[6] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0};

  double *q = new double[SIZE];
  double *r = new double[SIZE];
  double *s = new double[SIZE];

  double* const c = r;
  c = a;                        //#14: (1)Valid or (2)Invalid ?

  c[0] = 15.0;                  //#15: (1)Valid or (2)Invalid ?

  const double* const b = q;
  b = a;                        //#16: (1)Valid or (2)Invalid ?

  b[0] = 17.0;                  //#17: (1)Valid or (2)Invalid ?

 const double* d = s;
  d = a;                        //#18: (1)Valid or (2)Invalid ?

  d[0] = 19.0;                  //#19: (1)Valid or (2)Invalid ?

}
```

KEY

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

b is a constant pointer to a constant target neither it nor its target can be re-assigned.

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

III. Class Basics

Assume the following class declaration and implementation:

```
class GasTank {
private:
   bool   cap;   //true = cap closed
   float  gals;  //number of gallons
public:
   GasTank();
   GasTank(bool  lid, float level);
   void  OpenCap ();
   void  CloseCap();
   float Capacity();
   void  Pump  (float amount);
   void  Siphon(float amount);
   ~GasTank();
};

GasTank:: GasTank (){
   cap  = true;
   gals = 0.0F;
}

GasTank:: GasTank (bool  lid,
                   float level){
   cap  = lid;
   gals = level;
}

GasTank:: ~GasTank () { }
```

```
void GasTank:: OpenCap() {
   cap = false;
}

void GasTank:: CloseCap() {
   cap = true;
}

float GasTank:: Capacity () {
   return( gals );
}

void GasTank:: Pump (float amount) {
   OpenCap();
   gals += amount;
   CloseCap();
}

void GasTank:: Siphon (float amount) {
   OpenCap();
   gals -= amount;
   CloseCap();
}
```

KEY

The constructor executes on object creation. Pump() executes and calls OpenCap() & CloseCap(). The destructor executes upon yje object going out of scope.

Circle the number of the best answer to each question:

#20 Given the `main` function at the right, how many functions (not counting `main` itself), would be executed by the code?

```
void main( ) {
  GasTank FuelTank;

  FuelTank.Pump(20.0F);

}//end main()
```

(1) 1 (2) 2 (3) 3 (4) 4

(5) 5 (6) 6 (6) None of the above

#21 Given that a `bool` variable is stored in 1 byte and a `float` variable is stored in 4 bytes, how many bytes of dynamic memory is allocated by the code in `main`() in the previous problem? (Count carefully.)

(1) 0 (2) 2 (3) 9 (4) 13

(5) 5 (6) 15 (7) None of the above

The only memory in main() above is static memory.

#22 How many of the member functions in the `GasTank` class should have been declared as `const` member functions?:

(1) 1 (2) 2 (3) 3 (4) 4
(5) 5 (6) 6 (7) 7 (8) 0

Only the capacity member function does not change any of the private data members.

#23 How many constructor members does the `GasTank` class declaration contain?

(1) 1 **(2) 2** (3) 3
(4) 4 (5) 0 (6) None of the above

GasTank contains a default (parameterless) constructor and a parameterized constructor.

#24 What do the statements at the right accomplish:

```
void main( ) {
 GasTank CarTank(true, 20.0F);
 CarTank.Siphon(20.0F);
}//end main()
```

CarTank is instantiated with 20 gal.s of fuel and a closed cap. Siphon opens the cap, removes the 20 gal.s and closes the cap. This leaves the CarTank in an empty state the same as default constructed GasTank object.

(1) instructs the `GasTank` object `CarTank` to fully empty its tank.

(2) instructs the `GasTank` object `CarTank` to open its cap and discharge 20.0 gallons. *== 0.5 credit*

(3) instructs the `GasTank` object `CarTank` to close its cap and add 20.0 to its gallons.

(4) instructs the `CarTank` object `GasTank` to fully empty its tank.

(5) instructs the `CarTank` object `GasTank` to open its cap and discharge 20.0 gallons.

(6) instructs the `CarTank` object `GasTank` to close its cap and add 20.0 to its gallons.

(7) the statements contains a syntax error

(8) None of these

KEY

#25 What do the following statements accomplish:

```
bool EmptyWarning (GasTank tank); //prototype

   // in main ()
   GasTank CycleTank(true, 3.0F);            //Line 1
   if (EmptyWarning(CycleTank) )             //Line 2
         cout << "***Fuel Low***";

//Function Definition
bool EmptyWarning (const GasTank& tank) {    //Line 3
   return( tank.gals <= 3.0F );              //Line 4
}
```

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

(1) causes the `CycleTank` object to display a warning message

(2) causes the `GasTank` object to display a warning message

(3) generates a compiler error message on line 1

(4) generates a compiler error message on line 2

(5) generates a compiler error message on line 3

(6) generates a compiler error message on line 4

(7) None of these