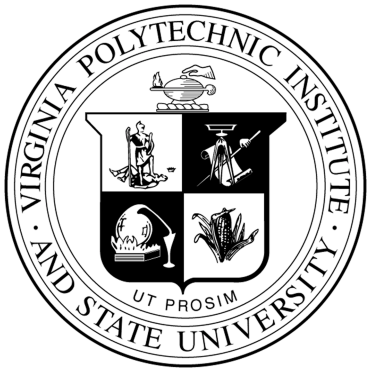


READ THIS NOW!

- Print your name in the space provided below. For Mr. Barnette's section code a group of '**1**' for Mr. McQuain's 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!

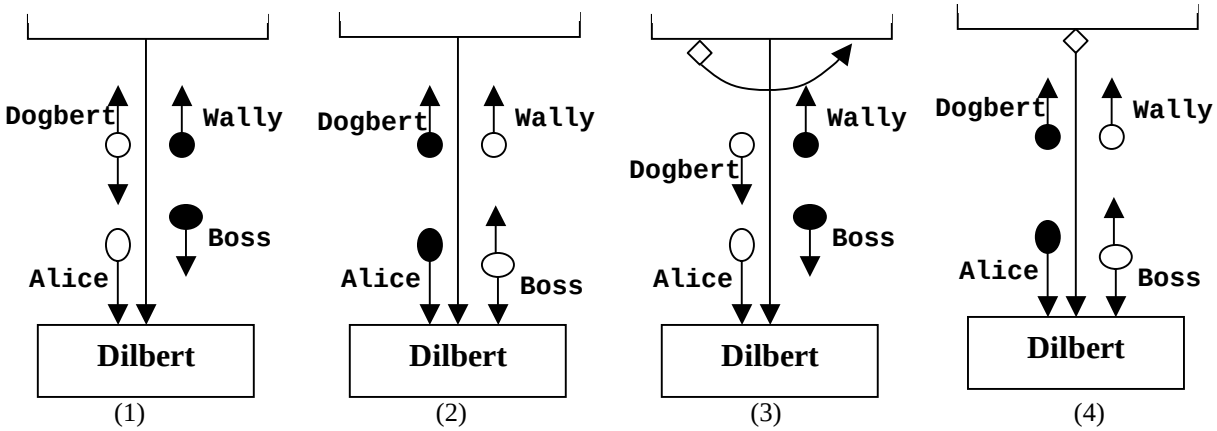
Print Name (Last, First) _____

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

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

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

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

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

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

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

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

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 [SIZE] a;
```

#4
```
b = &a[SIZE - 1];
```

#5
```
*&a[0] = y[1];
```

#6
```
*(a+1) = a[0];
```

#7
```
*a[SIZE-1] = *y[SIZE-1];
```

#8
```
y = &a[0];
```

#9 Identify the logical error that occurs in the statements:

(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 *p = new char[5];
char *q = p;
q[0]= 'D'; p[1]= 'Q';
q[2]= 'Q'; p[3]= '\0';
```

#10 Identify the logical error that 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 *q = new char[5];
q[0]= 'A'; q[1]= '\0';
delete [] q;
q[0] = 'C';
```

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

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

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

for (int i = 0; i < SIZE; i++)
      x[i] = i;
y = x;
y = y + 2;
cout << " y = " << y << endl;
```

(1) 00002002 (2) 00002004 (3) 00002008

(4) 1 (5) 2 (6) None of the above

Consider the following code:

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

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

  GetMem( a, SIZE, SIZE);

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

  delete [] a;
}
```

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

#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()` does 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 size parameter must not also be passed as the init parameter to prevent it from being corrupted when the size alias is decremented.

(3) the integer pointer parameter, a, must be passed as a constant pointer 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 parameter to prevent the changes made by `GetMem()` from being inadvertently lost and an illegal access occurring when the function returns.

(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 f1(const int* const arg);
void f2(int* const arg);
void f3(const int* arg);

void fn(int* p) {

  int *q = p;
  int *r = p;
  int *s = p;

  f1(q);

  f2(r);

  f3(s);

}

void f1(const int* const arg){
  int a[6] = {5, 4, 3, 2, 1, 0};

  arg = a;                    //#14: (1)Valid or (2)Invalid ?

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

void f2(int* const arg) {
  int a[6] = {5, 4, 3, 2, 1, 0};

  arg = a;                    //#16: (1)Valid or (2)Invalid ?

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


void f3(const int* arg) {
  int a[6] = {5, 4, 3, 2, 1, 0};

  arg = a;                    //#18: (1)Valid or (2)Invalid ?

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

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

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

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

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

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

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

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

Circle the number of the best answer to each question:

#20 What does the following statement accomplish:

```
GasTank FuelTank;
```

(1) define an instance of the class `FuelTank.`

(2) define an instance named `GasTank` of a class `FuelTank` with a closed cap and which is empty.

(3) define an instance named `FuelTank` of a class `GasTank` with a closed cap and which is empty.

(4) define an instance named `GasTank` of a class `FuelTank` with unknown status.

(5) define an instance named `FuelTank` of a class `GasTank` with unknown status.

(6) None of these

#21 What does the following statement accomplish:

```
GasTank CarTank(true, 20.0F);
```

(1) define an instance of the class `CarTank.`

(2) define an instance named `GasTank` of a class `CarTank` with unknown status.

(3) define an instance named `CarTank` of a class `GasTank` with unknown status.

(4) define an instance named `GasTank` of a class `CarTank` with a closed cap and 20 gallons.

(5) define an instance named `CarTank` of a class `GasTank` with a closed cap and 20 gallons.

(6) None of these

#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

#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

#24 What do the following statements accomplish:

```
GasTank TruckTank(true, 5.0F);
TruckTank.Siphon(5.0F);
```

(1) instructs the `GasTank` object `TruckTank` to fully empty its tank.
(2) instructs the `GasTank` object `TruckTank` to open its cap and discharge 5.0 gallons.
(3) instructs the `GasTank` object `TruckTank` to close its cap and add 5.0 to its gallons.
(4) the statement contains a syntax error
(5) None of these

#25 What do the following statements accomplish:

```
bool EmptyWarning (GasTank tank);

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


bool EmptyWarning (GasTank tank) {
   return( tank.Capacity() < 3.0F );
}
```

(1) causes the `CycleTank` object to display a warning message
(2) causes the `GasTank` object to display a warning message
(3) does not cause the `CycleTank` object to display a warning message
(4) does not cause the `GasTank` object to display a warning message
(5) None of these