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 B** 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 33 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. Pointers

Use the responses:

   (1) Valid   (2) Invalid

for the next 6 questions (#1 - #6). Considering each numbered question statement in the function below separately, determine whether each statement would be valid or invalid:

Assume the following function declaration:

```
void fn( char* const a) {

  char s[3] = {'a', 'b', 'c'};

  const char* const b = s;

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

  *b = 'b';              //#2: (1)Valid or (2)Invalid ?

  const char* c = s;

  c = a;                 //#3: (1)Valid or (2)Invalid ?

  *c = 'c';              //#4: (1)Valid or (2)Invalid ?

  char* const d = a;

  d = a;                 //#5: (1)Valid or (2)Invalid ?

  *d = 'd';              //#6: (1)Valid or (2)Invalid ?

}
```

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

```
      const int SIZE = 10;
      int* a; int* b;

      a = new int[SIZE];  // assume allocation starts at address 00001000

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

      b = a;
      b = b + SIZE;
      cout << " b = " << *b << endl;
```

(1) 00001000   (2) 00001004   (3) 0

(4) 1   (5) 10   (6) None of the above

Consider the following code:

```
void resize (const int*& ray,          //resize actual array to dimension now
            int then, int now);       void resize (const int*& ray, int then,
                                                   int now)
const int SIZE = 10;                   {
void main() {                            int *tmp, *p, *q;
int* a;                                  int i;

  a = new int[SIZE];
                                         p = tmp = new int[now]; //get new array
  for (int i =0; i < SIZE; i++)          for (i=0, q=ray; i<then; i++, p++, q++)
     a[i] = i;                              *p = *q;          //copy from ray to tmp
                                         delete [SIZE] ray; //deallocate ray
  resize(a, SIZE, SIZE/2);               ray = tmp;          //point ray to new
}                                       }
```

#8   In the code above, how is the dynamic array int pointer variable a being passed to the resize() 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

#9   Which of the following statements best describe the result/effect of the call to the resize() function from the
        main() function?

   (1)   the resize() function will correctly allocate a new array, copy the old contents of a into it, remove the memory
          previously allocated to a and reassign a to point to the new array.

   (2)   the call to the resize function will result in an array bounds violation when array a is reassigned to the new
          allocated array.

   (3)   the call to the resize function will result in an array bounds violation when array a's contents is copied to the new
          allocated array.

   (4)   the call to the resize function will result in the newly allocated array containing locations that have not been
          initialized.

   (5)   none of the above

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 7 questions (#10 - #16). Considering each statement below **independently**, determine whether each statement would compile (not link) without errors after the statement:

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

#10
```
b = &y[SIZE];
```

#11
```
*a = *y;
```

#12
```
(*a).[SIZE-1] = (*y).[SIZE-1];
```

#13
```
&y = &a;
```

#14
```
y = NULL;
```

#15
```
a = new int[SIZE];
```

#16
```
y = new int[SIZE];
```

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

(1) Alias pointer exists
(3) Illegal memory address reference
(5) Undefined pointer dereferenced

(2) Dangling Reference exists
(4) Memory garbage exists
(6) No logical error occurs

```
char *p = new char[5];
strcpy(p, "CPP");

p = &p[0];
```

#18 Identify the logical error that occurs in the code fragment:

(1) Alias pointer exists
(3) Illegal memory address reference
(5) Undefined pointer dereferenced

(2) Dangling Reference exists
(4) Memory garbage exists
(6) No logical error occurs

```
char *q = new char[5];
strcpy(q, "BSS");
char *r;

r = &q[0];
```

II. Design Representation

Consider the following partial Structure Chart diagram below:
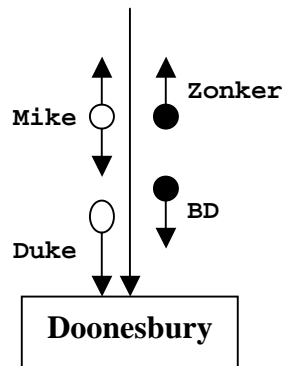
Do not make any assumption about variables that are not shown on the chart. Given the following variables definitions:

```
bool    BD,    Zonker;
int     Mike, Duke;
```

Which of the following incomplete function calls and function heading code below for `Doonesbury()` correctly models the diagram above, (more than 1 may be a valid model:

#19
```
Doonesbury(Mike, Zonker, Duke, BD);
if (Zonker)
    //code under control of if
```

```
void Doonesbury(int& Mike, bool& Zonker,
        const int& Duke, bool  BD) {
    if (BD)
        //code under control of if
```

    (1) – correct
    (2) – incorrect

#20
```
Doonesbury(Mike, Zonker, Duke, BD);
if (Mike > 0)
    //code under control of if
```

```
void Doonesbury(int& Mike, bool Zonker,
        int  Duke, const bool&  BD) {
    if (Mike < 0)
        //code under control of if
```

    (1) – correct
    (2) – incorrect

#21
```
Doonesbury(&Mike, &Zonker, Duke, BD);
if (!Zonker)
    //code under control of if
```

```
void Doonesbury(int* Mike, bool* Zonker,
        int Duke, const bool&  BD) {
    if (!BD)
        //code under control of if
```

    (1) – correct
    (2) – incorrect

#22
```
Doonesbury(*Mike, *Zonker, Duke, BD);
if (Zonker == NULL)
    //code under control of if
```

```
void Doonesbury(int& Mike, bool& Zonker,
        const int Duke, const bool  BD) {
    if (BD != NULL)
        //code under control of if
```

    (1) – correct
    (2) – incorrect

 III. Class Basics

Assume the following class declaration and implementation:

```
class ShotGun {
private:
   bool  safety;   //true - cannot fire
   int   rounds;   //number of shots
public:
   ShotGun(bool safe=true, int ammo=0);
   void safetyon();
   void safetyoff();
   bool ready();
   void load (int shells);
   void eject(int shells);
   int  bullets();
};

ShotGun:: ShotGun (bool safe, int ammo)
{
   safety = safe;
   rounds = ammo;
}

void ShotGun:: safetyon () {
   safety = true;
}
```

```
void ShotGun:: safetyoff () {
   safety = false;
}

bool ShotGun:: ready () {
   return((!safety) && (rounds > 0));
}

void ShotGun:: load (int shells) {
   rounds += shells;
}

void ShotGun:: eject (int shells) {
   rounds -= shells;
}

int ShotGun:: bullets () {
   return rounds;
}
```

Circle the number of the best answer to each question:

#23   What does the following statement accomplish: `ShotGun Browning(false, 6);`

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

   (2) define an instance named `Browning` of a class `ShotGun` with unknown status.

   (3) define an instance named `ShotGun` of a class `Browning` with unknown status.

   (4) define an instance named `ShotGun` of a class `Browning` with 6 possible shots and ready to fire.

   (5) define an instance named `Browning` of a class `ShotGun` with 6 possible shots and ready to fire.

   (6) None of these

#24   What does the following statement accomplish: `ShotGun Remington;`

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

   (2) define an instance named `Remington` of a class `ShotGun` with unknown status.

   (3) define an instance named `ShotGun` of a class `Remington` with unknown status.

   (4) define an instance named `ShotGun` of a class `Remington` with 0 possible shots and Not ready to fire.

   (5) define an instance named `Remington` of a class `ShotGun` with 0 possible shots and Not ready to fire.

   (6) None of these

#25   How many of the member functions in the `ShotGun` class should have been declared as `const` member functions?:
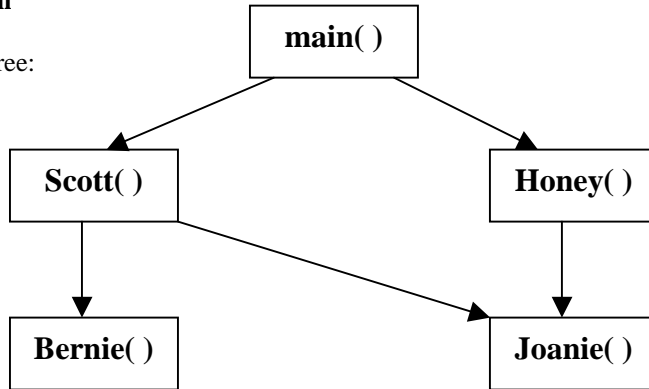
```
(1) 1           (2) 2         (3) 3         (4) 4
(5) 5           (6) 6         (7) 7         (8) 0
```

#26   How many default constructors does the above class declaration contain?

   (1) 1

   (2) 2

   (3) 3

   (4) 4

   (5) 0

IV. Separate Compilation

Consider the function call tree:

Assume that the software system is to be decomposed for compilation into three separate source files main.cpp, Scott.cpp, and Honey.cpp, and accompanying header files of the same names. The function definitions are to be placed in the various cpp files as shown below along with the corresponding code for the files.

FN definition locations

Definition for:	Goes in:
main( )	main.cpp
Joanie( )	main.cpp
Scott( )	Scott.cpp
Bernie( )	Scott.cpp
Honey( )	Honey.cpp

Scott separate compilation unit

```
//Scott.h
void Scott ( /* parameters */ );
```
```
//Scott.cpp
#include "Scott.h"
void Bernie( /* parameters */ );

void Scott ( /* parameters */ ){
// Scott's code
   Bernie();
   Joanie();
}

void Bernie ( /* parameters */ ){
// Bernie's code

}
```

Honey separate compilation unit

```
//Honey.h
void Honey ( /* parameters */ );
```
```
//Honey.cpp
#include "Honey.h"

void Honey ( /* parameters */ ){
// Honey's code
   Joanie ( /* parameters */ );
}
```

main separate compilation unit

```
//main.h
/* main declarations */
```
```
//main.cpp
#include "main.h"
void Joanie ( /* parameters */ );

void main() {

   Scott ( /* parameters */ );
   Honey ( /* parameters */ );

}

void Joanie ( /* parameters */ ){
// Joanie's code

}
```

IV. Separate Compilation (continued)

Assume that there are no global type and no constant declarations, (and also no global variables of course). Answer the following questions with respect to the above compilation organization and the goals of achieving information hiding and restricted scope:

#27   Assuming the partial code above was completed and contained no syntax errors, if **only** "`Honey.cpp`" is **compiled** (not built) within Microsoft Visual C++, which of the following type of errors would occur:

```
(1) Compilation errors: missing Honey() prototype
(2) Compilation error: undeclared identifiers 'Joanie'
(3) Linker Error: missing main function.
(4) No errors would be generated.
```

#28   Which of the following prototypes should be moved from its unit source file to the unit header file?

```
(1) void Honey ( /* parameters */ );       (2) void Bernie( /* parameters */ );

(3) void Scott  ( /* parameters */ );       (4) void Joanie ( /* parameters */ );
```

#29   In addition to the include directives listed above, where else should "`Honey.h`" be included?

```
(1) main.h         (2) main.cpp      (3) Scott.cpp
(4) Scott.h        (5) Honey.cpp     (6) nowhere else
```

#30   In addition to the include directives listed above, where else should "`Scott.h`" be included?

```
(1) main.h         (2) main.cpp      (3) Scott.cpp
(4) Honey.h        (5) Honey.cpp     (6) nowhere else
```

#31   In addition to the include directives listed above, where else should "`main.h`" be included?

```
(1) main.h         (2) main.cpp      (3) Honey.cpp
(4) Honey.h        (5) Scott.h       (6) nowhere else
```

#32   In how many different files (source and header) should the `#include "main.h"` directive occur?

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

#33   The name of the linker program that is invoked automatically by the Microsoft Visual C++ development environment is :

```
(1) cl
(2) ld
(3) ln
(4) link
(5) none of the above
```