READ THIS NOW!

- Print your name in the space provided below. Code Form A on your Opscan. Check your SSN and Form Encoding!
- 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, \[ 1704 \text{ (integer)}, \quad 1704.0 \text{ (real)}\].
- 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
1. **Class Pointers**

For the following 4 questions, assume the following declarations:

```cpp
class Article { //forward declaration
class Article {  
private:
   Article* bond;
   bool fact;
public: //member functions
   void statement();
   ~Article();
};

//inside the Article member function statement
Article* that = new Article;
that->fact = true;
that->bond = that;
Article piece = *that;
piece.bond->fact = false;
piece.bond->bond = &piece;
that = &piece;
that->bond->bond = that;
```

1. From inside the same member function as the above code, what is the **type**, (not value), of the expression at the right: `that->bond->bond`

   1) NULL  
   2) Article  
   3) Article*  
   4) bond  
   5) bond*  
   6) None of the above

2. From inside the same member function as the above code, which of the following statements could be used to change the `Article` object containing a false fact to a true fact (after the last statement above)?

   1) `piece.fact = true;`  
   2) `that->fact = true;`  
   3) `that->bond->fact = true;`  
   4) `piece->fact = true;`  
   5) `piece->bond->fact = true;`  
   6) None of the above

3. From inside the same member function as the above code, immediately before the function terminates, how many `Article` objects can be accessed?

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

4. Considering just the code above, after the member function, `statement()`, has completed execution, (i.e. went out of scope), how many `Article` objects would the destructor be executed upon?

   1) 1  
   2) 2  
   3) 3  
   4) 4  
   5) 0  
   6) None of the above
II. Linked List Class Manipulation

Consider the linked list class and list node declarations given below:

```cpp
class ItemType {
private:
    int Value;
public:
    ItemType();
    ItemType(int newValue);
    void setValue(int newValue);
    int getValue() const;
};
```

```cpp
class LinkNode {
private:
    ItemType Data;
    LinkNode* Next;
public:
    LinkNode();
    LinkNode(ItemType newData);
    bool setNext(LinkNode* newNext);
    bool setData(ItemType newData);
    ItemType getData() const;
    LinkNode* getNext() const;
};
```

LinkNode *Head, *P, *Q;

Assume that the member functions above have been implemented correctly to carry out their intended task. Also, assume that operations have been executed to create the initial list structure below:

For the next 4 questions, select missing statements for the client, (not class), code segment below to transmogrify the above list into the list shown below:

```cpp
LinkNode * x = Head; //initialize x to first node
LinkNode * y = ________________; //5. //initialize y to last node
LinkNode* t;
if (x != NULL && y != NULL && x != y) { //check trivial cases
    do {
        for (t=x; (t!=NULL && t->getNext()!= y); ) //hmmm what is this for doing?
            t = t->getNext();
        //swap *x & *y
        int s = ________________; //6. //assign s to *x
        x-> ________________; //7. //assign *x to *y
        y->setData(s);
        x = x->getNext(); //increment x
        y = ________________; //8. //decrement y
    } while ( x->getData().getValue() > y->getData().getValue() );
} //if
```

![Diagram of the initial list structure](attachment:initial_list.png)

![Diagram of the final list structure](attachment:final_list.png)
II. Linked List Class Manipulation (continued)

Select from the possible answers for the 4 questions given on the previous page.

<table>
<thead>
<tr>
<th></th>
<th>1) Q-&gt;Next-&gt;Next</th>
<th>2) Q-&gt;getNext()-&gt;getNext()</th>
</tr>
</thead>
<tbody>
<tr>
<td>3)</td>
<td>*x</td>
<td></td>
</tr>
<tr>
<td>4)</td>
<td>x-&gt;getData()</td>
<td></td>
</tr>
<tr>
<td>5)</td>
<td>x-&gt;getData().getValue()</td>
<td></td>
</tr>
<tr>
<td>6)</td>
<td>setData(y-&gt;getData().getValue())</td>
<td></td>
</tr>
<tr>
<td>7)</td>
<td>setData(y-&gt;Data.Value)</td>
<td></td>
</tr>
<tr>
<td>8)</td>
<td>y-&gt;getNext()</td>
<td></td>
</tr>
<tr>
<td>9)</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>10)</td>
<td>t-&gt;getNext()</td>
<td></td>
</tr>
</tbody>
</table>
III. Separate Compilation

For the next three questions, consider a C++ program composed of three .cpp files and three corresponding header files, as shown below. (the name of the file is in the first comment line of the file). All function calls are shown, as are all type declarations, function prototypes and some include directives. In the source and header files, there should be only one physical occurrence of a function prototype, and one physical occurrence of a type declaration. Do not assume that any preprocessor directives are used but not shown.

```cpp
// main.h
class mClass {
    // ...
};

// main.cpp
#include "main.h" //Line 9
#include "Exam2.h" //Line 10

// ...
void main() {
    Class2 C;
    Exam2(C);
    // ...
} //end main

// ...
// mClass member Functions

// Exam2.h
void Exam2(Class2& C2obj);

// Exam2.cpp
#include "Exam2.h"
// ...
void Exam2(Class2& C2obj) {
    // ...
}

// Class2.h
#include "Class2.h" //Line 11

// ...
class Class2 {
    private:
        mClass MObj;
    // ...
};

// Class2.cpp
#include "Exam2.cpp"
// ...
// Class2 member Functions

// Class2.cpp
#include "Class2.h"
// ...
// Class2 member Functions
```

9. If the organization shown above is used, and no preprocessor directives are added, which of the following include directives should replace the underscores at Line 9 above so that main.cpp can be successfully compiled?

1) Nothing.
2) #include "main.h"
3) #include "Exam2.h"
4) #include "Class2.h"
5) #include "Exam2.cpp"
6) #include "Class2.cpp"
7) None of these.

10. If the organization shown above is used, and no preprocessor directives are added, which of the following include directives should replace the underscores at Line 10 above so that main.cpp can be successfully compiled?

1) Nothing.
2) #include "main.h"
3) #include "Exam2.h"
4) #include "Class2.h"
5) #include "Exam2.cpp"
6) #include "Class2.cpp"
7) None of these.

11. If the organization shown above is used, and no preprocessor directives are added, which of the following include directives should replace the underscores at Line 11 above so that Class2.cpp can be successfully compiled?

1) Nothing.
2) #include "main.h"
3) #include "Exam2.h"
4) #include "Class2.h"
5) #include "Exam2.cpp"
6) #include "Class2.cpp"
III. Separate Compilation (continued)

Consider the function call tree:

![Function Call Tree]

Assume that the software system is to be decomposed for compilation into three separate source files: `main.cpp`, `Sid.cpp`, and `Pitr.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.

<table>
<thead>
<tr>
<th>FN definition locations</th>
<th>Scott separate compilation unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition for:</td>
<td>//Pitr.h</td>
</tr>
<tr>
<td>main( )</td>
<td>void Pitr ( /* parameters */ );</td>
</tr>
<tr>
<td>Sid( )</td>
<td>// Pitr.cpp</td>
</tr>
<tr>
<td>#include “Pitr.h”</td>
<td>void Erwin( /* parameters */ );</td>
</tr>
<tr>
<td>void Pitr ( /* parameters */ ) {</td>
<td></td>
</tr>
<tr>
<td>// Pitr’s code</td>
<td>Erwin();</td>
</tr>
<tr>
<td>Erwin();</td>
<td>Stef();</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>Erwin( )</td>
<td>void Erwin ( /* parameters */ );</td>
</tr>
<tr>
<td>#include “Erwin.h”</td>
<td>// Erwin’s code</td>
</tr>
<tr>
<td>void Erwin ( /* parameters */ ) {</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>Erwin( /* parameters */ );</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sid separate compilation unit</th>
<th>main separate compilation unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>//Sid.h</td>
<td>//main.h</td>
</tr>
<tr>
<td>void Sid ( /* parameters */ );</td>
<td>/* main declarations */</td>
</tr>
<tr>
<td>int greg;</td>
<td></td>
</tr>
<tr>
<td>// Sid.cpp</td>
<td></td>
</tr>
<tr>
<td>#include “Sid.h”</td>
<td></td>
</tr>
<tr>
<td>void Stef ( /* parameters */ );</td>
<td></td>
</tr>
<tr>
<td>void Sid ( /* parameters */ ){</td>
<td></td>
</tr>
<tr>
<td>// Sid’s code</td>
<td></td>
</tr>
<tr>
<td>Stef( /* parameters */ );</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>void Stef ( /* parameters */ ){</td>
<td></td>
</tr>
<tr>
<td>// Stef’s code</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>//main.cpp</td>
<td></td>
</tr>
<tr>
<td>#include “main.h”</td>
<td></td>
</tr>
<tr>
<td>#include “Sid.h”</td>
<td></td>
</tr>
<tr>
<td>void main() {</td>
<td></td>
</tr>
<tr>
<td>Sid ( /* parameters */ );</td>
<td></td>
</tr>
<tr>
<td>Pitr ( /* parameters */ );</td>
<td></td>
</tr>
<tr>
<td>Erwin( /* parameters */ );</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>
III. Separate Compilation (continued)

Assume that there are no global type and no global 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:

12. Assuming the partial code above was completed and contained no syntax errors, if only “Pitr.cpp” is compiled (not built) within Microsoft Visual C++, which of the following type of errors would occur:

   1) Compilation error C2065: Erwin : undeclared identifier
   2) Compilation error C2065: Stef : undeclared identifier
   4) No errors would be generated.

13. Which of the following prototypes should be moved from its unit source.cpp file to the unit header.h file?

   1) void Sid ( /* parameters */ );
   2) void Pitr( /* parameters */ );
   3) void Erwin ( /* parameters */ );
   4) void main ( );

14. In addition to the include directives listed above, where else should “Pitr.h” be included?

   (1) main.h   (3) Sid.h   (5) Pitr.h
   (2) main.cpp (4) Sid.cpp (6) nowhere else

15. In addition to the include directives listed above, where else should “Sid.h” be included?

   (1) main.h   (3) Pitr.cpp   (5) nowhere else
   (2) Pitr.h   (4) Sid.h

16. Assume all the code above was completed and contains no syntax or compilation errors. Further, assume that all of the header files have appropriate conditional compilation directives surrounding their contents. When the project containing the files is built within Microsoft Visual C++, which of the following linker errors would occur:

   1) error LNK2015: multiple definitions for identifier ‘Stef’
   2) error LNK2015: multiple definitions for identifier ‘Erwin’
   3) Pitr.obj : error LNK2005: "int greg" already defined in Sid.obj
   4) No errors would be generated.
IV. Object Manipulations

Assume the following class declaration and implementation:

```cpp
class IlliadUF {
private:
  bool* unix;
public:
  IlliadUF();
  IlliadUF(bool LordCrud);
  IlliadUF(const IlliadUF& Mike);
  bool getTF() const;
  void setTF(bool truth);
  ~IlliadUF();
};
IlliadUF::IlliadUF () {
  unix = new bool(true);
}
IlliadUF::IlliadUF (bool LordCrud) {
  unix = new bool(LordCrud);
}
IlliadUF::IlliadUF (const IlliadUF& Mike) {
  unix = new bool(*Mike.unix);
}
IlliadUF::~IlliadUF () {
  delete unix;
}

bool IlliadUF::getTF() const {
  return(*unix);
}
void IlliadUF::setTF(bool truth) {
  *unix = truth;
}
IlliadUF::IlliadUF (const IlliadUF& Mike) {
  unix = new bool(*Mike.unix);
}
IlliadUF::~IlliadUF () {
  delete unix;
}
```

Given the following code:

```cpp
void DustPuppy(IlliadUF Chief);

void main() {
  IlliadUF Miranda(false), AJ = Miranda;
  IlliadUF Cobb, Hillary(true);

  DustPuppy(Miranda);
  Cobb = Hillary;
  Hillary.setTF(!Hillary.getTF());

  cout << boolalpha; //stream modifier to output true/false for bools
  cout << "Contents of Miranda is:" << Miranda.getTF() << endl; //LINE 1
  cout << "Contents of AJ is:" << AJ.getTF() << endl; //LINE 2
  cout << "Contents of Cobb is:" << Cobb.getTF() << endl; //LINE 3
  cout << "Contents of Hillary is:" << Hillary.getTF() << endl; //LINE 4
}

void DustPuppy(IlliadUF Chief) { Chief.setTF(!Chief.getTF()); }
```

For the next 4 questions, select your answers from the following:

1) true 
2) false 
3) Execution Error 
4) None of these

17. What bool value is output by the call `Miranda.getTF()` in LINE 1 above?

18. What bool value is output by the call `AJ.getTF()` in LINE 2 above?

19. What bool value is output by the call `Cobb.getTF()` in LINE 3 above?

20. What bool value is output by the call `Hillary.getTF()` in LINE 4 above?
21. In the above code, immediately before `main()` goes out of scope, what is the total number of `IlliadUF` objects that has been dynamically allocated, (include in the count any that have been allocated and destructed).

   (1) 1  (3) 3  (5) 5  (7) 7  (9) 0
   (2) 2  (4) 4  (6) 5  (8) 8  (10) None of the above

V. Recursion

Assume that the `LinkNode` and `LinkList` classes discussed in class have been implemented correctly and are available for use. The `LinkList` and `LinkNode` interfaces are given below:

```cpp
#include "LinkNode.h" // for node declaration
#include "Item.h"

class LinkList {
private:
    LinkNode* Head; // points to head node in list
    LinkNode* Tail; // points to tail node in list
    LinkNode* Curr; // points to "current" node
public:
    LinkList(); //constructor
    LinkList(const LinkList& Source);
    LinkList& LinkList::operator=(const LinkList& otherList);
    ~LinkList(); //destructor
    bool isEmpty() const;
    bool inList() const;
    bool PrefixNode(const Item& newData);
    bool Insert(const Item& newData);
    bool Advance();
    void gotoHead();
    void gotoTail();
    bool DeleteCurrentNode();
    bool DeleteValue(const Item& Target);
    Item getCurrentData() const;
    void setCurrentData(const Item& newData);
};
```

Assume that the `Item` type has been typedef’d to be equivalent to an `int` and that each node of the list holds one digit of an integer number. If the list stores a date then we might wish to code a recursive function to determine if the date is a palindrome. A palindrome is something that is the same forwards as backwards. For example, the date 10 02 2001 would be a palindromic date, (Europeans list the day first, so for them 20 02 2002 would be a palindromic date).

Given the following, incomplete, palindrome list class member functions:

```cpp
bool LinkList::ListPalindrome() {
    //setup function for Palindrome which does the real work
    if ( ! isEmpty() ) {
        gotoHead();
        bool palin = Palindrome();
        for (; (Tail->getNext() != NULL); Tail = Tail->getNext() ) ;
        return ( palin );
    } else
        return (false);
}
```

//LinkNode.h
#include "Item.h"
class LinkNode {
private:
    Item Data; //data "capsule"
public:
    LinkNode* Next; //pointer next node
    LinkNode();
    LinkNode(const Item& newData);
    void setData(const Item& newData);
    void setNext(LinkNode* const newNext);
    Item getData() const;
    LinkNode* getNext() const;
};
} // ListPalindrome

bool LinkList::Palindrome() {
    LinkNode* tmp;
    if (_________________) //22.
        return true;
    if (_______________) //23.
        return (false);
    Advance();
    for (tmp = Curr; tmp->getNext() != Tail; tmp = tmp->getNext()) ;//null for
    __________________; //24.
    return ( Palindrome() );
} // Palindrome

22. Select from the missing statements below to correctly fill in the blank for line numbered //22. in the above code to correctly satisfy the first base case of the Palindrome function?
1) Head == NULL  
2) Head == Tail  
3) Curr == NULL  
4) Curr == Head  
5) Curr == Tail  
6) None of the above

23. Select from the missing statements below to correctly fill in the blank for line numbered //23. in the above code to correctly satisfy the second base case of the Palindrome function?
1) Head->getData() == Tail->getData()  
2) Head->getData() != Tail->getData()  
3) getCurrentData() == Tail->getData()  
4) getCurrentData() != Tail->getData()  

24. Select from the missing statements below to correctly fill in the blank for line numbered //24. in the above code to correctly setup the recursive call to the Palindrome function?
1) Head = tmp  
2) Curr = tmp  
3) Tail = tmp  
4) Head = Curr  
5) Tail = Curr  
6) None of the above

25. Which of the recursive problem solution methods is the Palindrome() function an example?
1) Tail (going up) recursion  
2) Head (going down) recursion  
3) Middle Decomposition  
4) Edges & Center Decomposition  
5) Backtracking  
6) None of the above