

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; code your section **group** number: Barnette 11:00 TuTh = 1; McQuain 10:00 MWF = 2; or McQuain 12:00 MWF = 3.
- 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 20 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. Class Pointers

For the following 2 questions, assume the following declarations:

```
class Nodule;
typedef Nodule* Linkage;
class Nodule {
private:
    int     element;
    Linkage link;
public: //member functions
    void foo();
};
```

```
//inside the Nodule member function foo
Nodule component;
component.element = 0;
component.link = NULL;
Linkage bond, tie = &component;
bond = new Nodule;
bond->element = 1;
tie->link = bond;
bond = new Nodule;
bond->element = 2;
bond->link = tie;
```

- From inside the same member function as the above code, what is the data type of the expression at the right: tie->link
 - 1) NULL
 - 2) Nodule
 - 3) Nodule*
 - 4) int
 - 5) struct*
 - 6) None of the above
- From inside the same member function as the above code, which of the following statements could be used to connect, (i.e., link), the Nodule containing the integer 1 to the Nodule containing the integer 2?
 - 1) bond->link = tie;
 - 2) component->link = bond;
 - 3) tie->link = bond;
 - 4) tie->link->link = bond;
 - 5) bond->link->link = tie;
 - 6) None of the above
- From inside the same member function as the above code, assuming the Nodules in the previous question are linked, what would be the data type of the expression at the right? tie->element->link
 - 1) NULL
 - 2) Nodule
 - 3) Nodule*
 - 4) int
 - 5) struct*
 - 6) None of the above
- Assuming the default (language supplied) destructor (i.e. no destructor has been explicitly implemented), consider just the code above, after the member function would complete execution, how many memory leaked Nodules would still exist in memory?
 - 1) 1
 - 2) 2
 - 3) 3
 - 4) 4
 - 5) 0
 - 6) None of the above

II. Linked List Class Manipulation

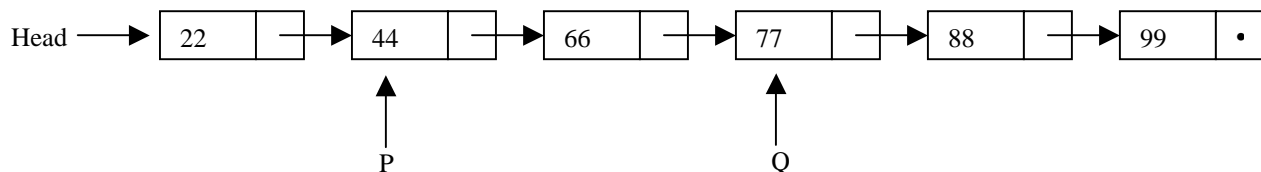
Consider the linked list class and list node declarations given below:

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

```
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. Given the initial list structure:



For the next 4 questions, determine what the execution of the given code fragment would do, assuming the list structure above as your starting point (for each question). Note – dangling references into the heap should be ignored, indicated by “??”. Choose from the possible answers given on the following page.

5.

```
LinkNode *R;
R = Q->getNext()->getNext();
R->setNext(P->getNext());
```
6.

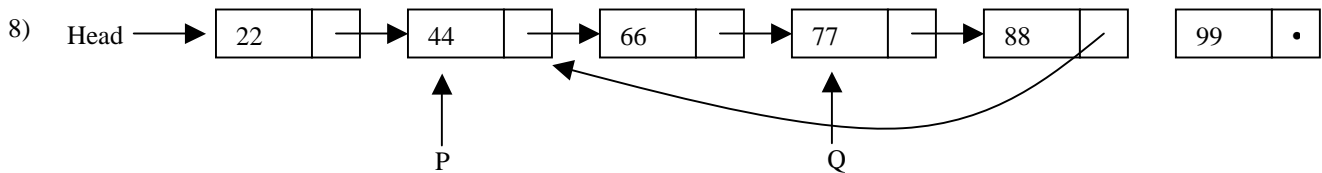
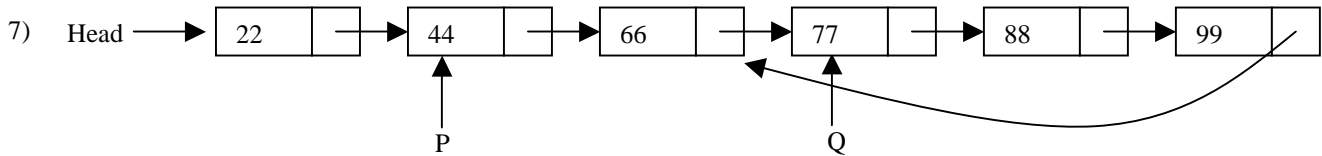
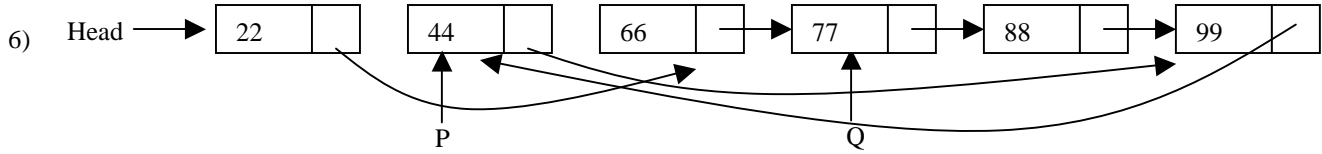
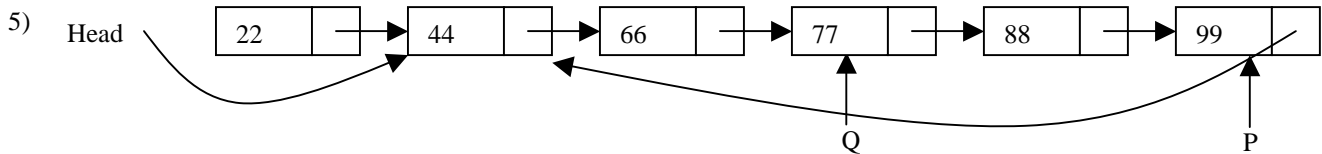
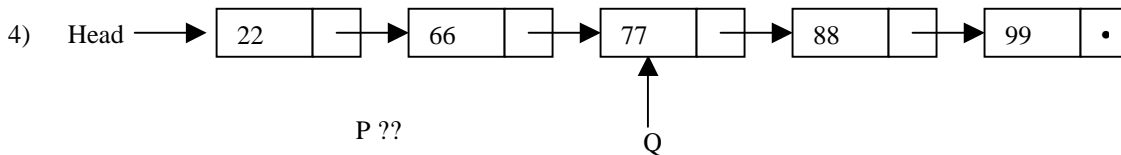
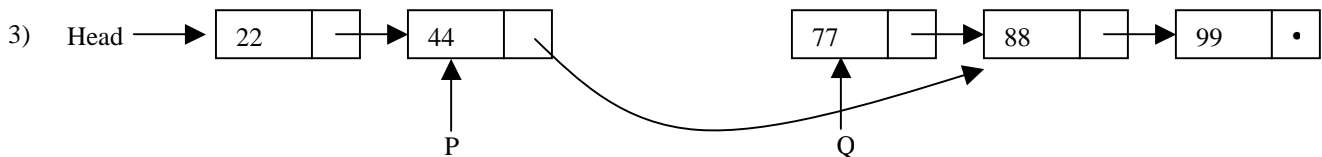
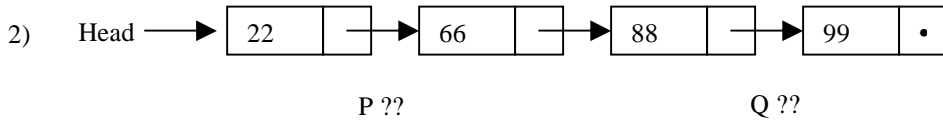
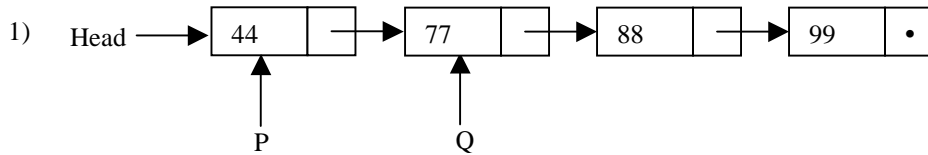
```
LinkNode *S=Head;
delete P->getNext();
S->setNext(Head->getNext());
P->setNext(Q->getNext());
```
7.

```
LinkNode *T=P;
while (T->getNext() != NULL) T = T->getNext();
Head->setNext(P->getNext());
P->setNext(T);
T->setNext(Head->getNext());
```
8.

```
LinkNode *Y, *X=Head;
for (int i=0; i<2; i++) {
    Y = X->getNext()->getNext();
    delete X->getNext();
    X->setNext() = Y;
    X = Y;
}
```

II. Linked List Class Manipulation *(continued)*

Select from the possible answers for the 4 questions given on the previous page. Question marks (??) indicate the pointer has an unknown, or invalid, value.



III. Command Line Arguments

Consider a program, `view2`, that provides side-by-side coordinated scrolling of two text files, (for comparison purposes). The two files are allowed to be optionally specified as command line arguments, (CLA) to the program, e.g.,

```
view2 FileA.txt FileB.txt
```

For the next two questions, choose from the following possible answers:

- | | | |
|----------------------|-------------------------|-------------------------|
| 1) <code>arg</code> | 4) <code>argc[0]</code> | 7) <code>argv[0]</code> |
| 2) <code>argc</code> | 5) <code>argc[1]</code> | 8) <code>argv[1]</code> |
| 3) <code>argv</code> | 6) <code>argc[2]</code> | 9) <code>argv[2]</code> |

9. The code below checks for the existence of the optional command line arguments. Select the answer to fill in the blank to carry out the indicated task correctly. All of the blanks are to be filled in by the same choice.

```
if ( _____ > 1) //CLAs present
    if ((_____ < 3) || (_____ > 3))//too few or too many files
        cout <<"usage: view2 FileA.txt FileB.txt"<<endl;
    else if (_____ == 3) // process CLAs
```

10. The code below is to check for the existence of the first file, (i.e., `FileA.txt`). Select the answer to fill in the blank to carry out the indicated task correctly. All of the blanks are to be filled in by the same choice.

```
// process CLAs
ifstream file1, file2;

file1.open( _____ ); //use ios::nocreate if using old headers
if (file1.fail())
    cout << "****Error, File: "<< _____ << "Does Not Exist"<<endl;
```

IV. Object Manipulations

11. When a copy constructor function is NOT implemented in a class which contains dynamic data, then a *shallow* copy would be performed in each of the following described execution points in a program except one. Identify at which instance a *shallow* copy would **NOT** be automatically performed?

- 1) When an object is returned by a function.
- 2) When an object is initialized to another object in a definition.
- 3) When an object is passed by const reference.
- 4) When an object is passed by value.
- 5) None of the above, (all of the above situations would result in a *shallow* copy).

12. (True or False) When a copy constructor function is implemented in a class which contains dynamic data, then the assignment operator should always be over-loaded for safety and completeness?

- 1) True
- 2) False

IV. Object Manipulations *(continued)*

Assume the following class declaration and implementation:

<pre>class State { private: char* Status; public: State(); State(char Status); char getState(); void setState(char Code); bool operator==(const Foo& Other); ~State(); }; State:: State() { Status = new char('\0'); } State:: State(char Status) { this->Status = new char(Status); }</pre>	<pre>int State::getState() { return (*Status); } void State::setState(char Code) { *Status = Code; } bool State::operator== (const State& Other) { return (*Status) == *(Other.Status); } State::~ ~State() { delete Status; }</pre>
---	---

Given the following code:

```
State Output(State out);
State p, q('$');
State s, r = p;

void main() {
    r.setState('@');

    cout << "State of p is:" << p.getState() << endl;           //LINE 1
    s = Output(q);
    cout << "State of s is:" << s.getState() << endl;           //LINE 2
}

State Output(State out) {
    cout << "State of out = " << out.getState() << endl;       //LINE 3
    return out;
}
```

For the next 3 questions, select your answers from the following, ignoring the quotes:

- | | |
|---------------------|--------------------|
| 1) '\0' (null char) | 4) Execution Error |
| 2) '\$' | 5) None of these |
| 3) '@' | |

13. What is output by the call `p.getState()` in LINE 1 above?
14. What is output by the call `s.getState()` in LINE 2 above?
15. What is output by the call `out.getState()` in LINE 3 above?

V. Recursion

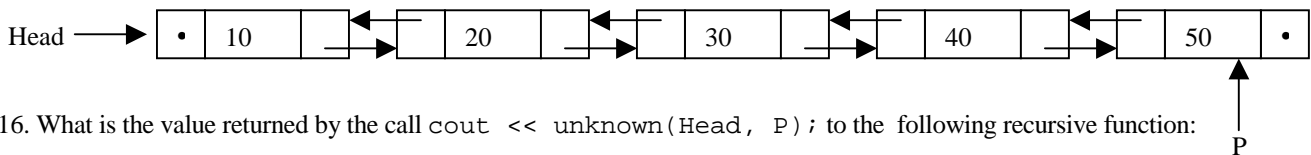
Consider the linked list class and list node declarations given below:

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

```
class LinkNode {
private:
    ItemType Data;
    LinkNode* Next, Prev;
public:
    LinkNode();
    LinkNode(ItemType newData);
    bool setNext(LinkNode* newNext);
    bool setPrev(LinkNode* newPrev);
    bool setData(ItemType newData);
    ItemType getData() const;
    LinkNode* getNext() const;
    LinkNode* getPrev() const;
};

LinkNode *Head, *P, *Q;
```

Assume that the member functions above have been implemented correctly to carry out their intended task. Given the initial list structure:



16. What is the value returned by the call `cout << unknown(Head, P);` to the following recursive function:

```
int unknown (LinkNode *R, LinkNode *S) {
if ((R == NULL) || (S == NULL)) return 0;
else if (R == S) return (R->getData()->getValue());
else {
    if ((R->getNext() == S) && (S->getPrev() == R))
        return (R->getData()->getValue() + S->getData()->getValue());
    else
        return (R->getData()->getValue() + S->getData()->getValue() +
                unknown(R->getNext(), S->getPrev()) );
}
}
```

- | | | |
|-------|-------|--------|
| 1) 10 | 4) 40 | 7) 120 |
| 2) 20 | 5) 50 | 8) 150 |
| 3) 30 | 6) 60 | 9) 0 |

17. The previous function is an example of what type of recursive problem solving strategy?

- | | |
|-----------------------------------|---------------------------------|
| 1) going up (tail) recursion | 4) edges & center decomposition |
| 2) going down (head) recursion | 5) backtracking |
| 3) middle decomposition recursion | 6) None of the above |

V. Recursion (*continued*)

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

18. In general, considering a correctly coded recursive function which for each call generates no more than one recursive call, how many times does the base case code execute? (Hint – consider the function in the previous question.)
- | | |
|--|---|
| 1) once at most | 4) equals the number of recursive calls |
| 2) at least twice | 5) never, there is no need |
| 3) as many times as the recursive code | 6) None of the above |

19. The following recursive function call, `Find(Element, IntArray, 0, Size)`, searches an array of integers for a desired element value and returns its index or negative one (-1) if the value is not in the array:

```
int Find( int Value, const int Array[], int Begin, int Dim ) {  
    if ( Begin >= Dim ) return -1;  
    if ( _____ )  
        return Begin;  
    else  
        return ( Find(Value, Array, Begin+1, Dim) );  
}
```

What should the missing condition in the if statement be?

- | | |
|---|---|
| 1) <code>Begin == 1</code> | 5) <code>Find(Value, Array, Begin, Dim) == Value</code> |
| 2) <code>Array[Begin] == Value</code> | 6) None of the above |
| 3) <code>Array[Begin-1] == Value</code> | |
| 4) <code>Array[Begin+1] == Value</code> | |

20. Consider the following recursive function:

```
int hmmm(int a, int b) {  
    if (a == 0) return (b + 1);  
    else if ((a!=0) && (b==0)) return (hmmm(a-1, 1));  
    else if ((a!=0) && (b!=0)) return (hmmm(a-1, hmmm(a, b-1) );  
}
```

//note the call within a call

What is returned from the call: `hmmm(2, 1)` ?

- | | | |
|------|------|------------------|
| 1) 1 | 4) 4 | 7) 7 |
| 2) 2 | 5) 5 | 8) 8 |
| 3) 3 | 6) 6 | 9) None of these |