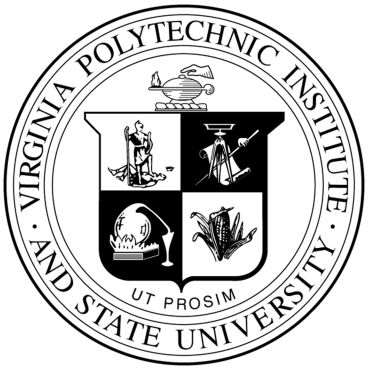


READ THIS NOW!

- 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 (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.

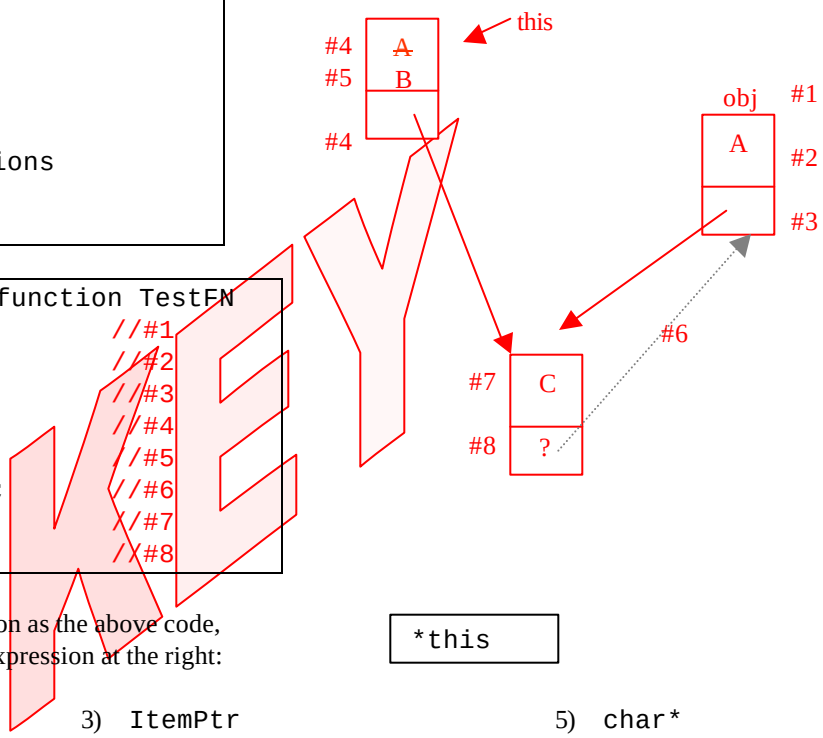
N. D. Barnette
signature

I. Class Pointers

For the following 4 questions, assume the following declarations:

```
class   Item; //forward declaration
typedef Item*  ItemPtr;
class   Item {
  private:
     ItemPtr  link;
     char     data;
  public: //member functions
      void TestFN();
};
```

```
//inside the Item member function TestFN
    Item obj;                    //#1
    obj.data = 'A';              //#2
    obj.link = new Item;         //#3
    *this = obj;                 //#4
    data = 'B';                  //#5
    obj.link->link = &obj;       //#6
    link->data = 'C';            //#7
    link->link = NULL;           //#8
```



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

```
*this
```

1) NULL
2) Item
3) ItemPtr
4) obj
5) char*
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 link, (i.e., point), the `Item` object containing the `char 'C'` to the `Item` object containing the `char 'B'`?

1) obj->link = this;
2) obj.link = link;
3) link->link = this;
4) this->link = link;
5) *this.link = &obj;
6) None of the above

3. From inside the same member function as the above code, what would be the type, (not value), of the expression at the right?

```
(*this).link->link
```

1) NULL
2) Item
3) ItemPtr
4) obj
5) char*
6) None of the above

4. Assume that the `Item` object which invokes the `TestFN` member function has not been allocated dynamically and that the value of the `link` pointer inside of it is `NULL` immediately before `TestFN` function is invoked. Note that the `Item` class relies upon the default (language supplied) destructor, (i.e. no destructor has been explicitly implemented). Considering just the code above, after the member function, `TestFN`, has completed execution, how many memory leaked `Item` objects would still exist, (orphaned), in memory?

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

II. Linked List Class Manipulation

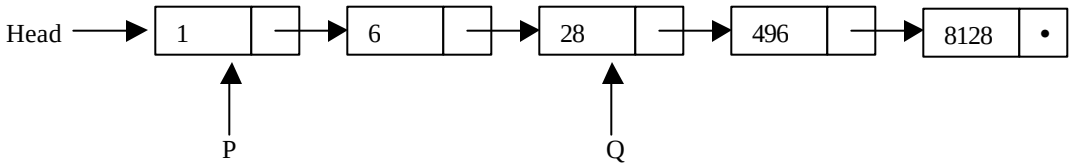
Consider the linked list class and list node declarations given below:

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

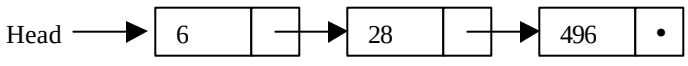
```
class LinkNode {
private:
   ItemType  Data;
   LinkNode* Next;
public:
   LinkNode();
   LinkNode(ItemType newData);
   bool setNext(LinkNode* newNext);
   bool setData(ItemType  newData);
   ItemType  getData() const;
   LinkNode* getNext() const;
};

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

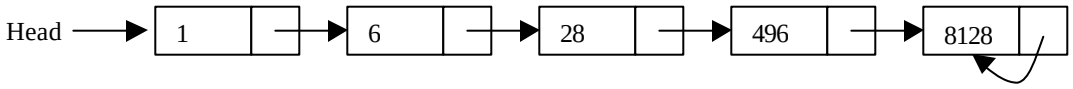
Assume that the member functions above have been implemented correctly to carry out their intended task. Given the initial list structure below:



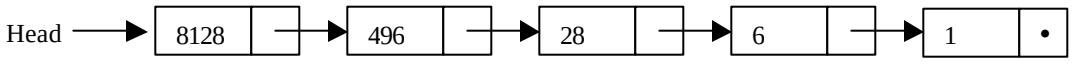
For the next 4 questions, select from the code segments on the following page, the segment that would transmogrify the above list into each of the lists shown below. Assume the list structure above as your starting point (for each question). Choose from the possible answers given on the following page.

5. 

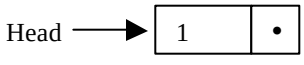
1

6. 

3

7. 

8

8. 

2

II. Linked List Class Manipulation (*continued*)

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

1)
```
LinkNode      *R=Head;
Head = R->getNext();
R->setNext(R->getNext()-
delete R;
R = Q->getNext();
Q->getNext()->
   setNext(R->getNext()-
delete R->getNext();
```
R->1
Head->6
1->28
del 1
R->496

496-> •

2)
```
LinkNode      *R=Head;
delete R->getNext();
R = P = Q = NULL;
Head->setNext(R);
```
R->1
del 6
RPQ-> •
1-> •

3)
```
LinkNode      *T=Head;
while (T->getNext()->get

    T = T->getNext();
} //while
T->getNext()->
   setNext(Q->getNext()-
T = T->getNext();
Q = T->getNext();
```
T->1

T->496

8128->8128
T->8128

4)
```
LinkNode      *T=NULL;
while (P->getNext()!= NUL
  T = P;
  P = P->getNext();
  delete T;
} //while
delete P;
P = Q = T = NULL;
Head->setNext(P);
```
del 1
del 6
del 28
del 496
P->8128
del 8128
PQT-> •

5)
```
for (int i=0; i<3; i++)
    P = P->getNext();
delete P->getNext();
P->setNext(NULL);
LinkNode *T=Head->getNex
Head->setNext(
  Head->getNext()->getNe
delete T;
T = Head->getNext();
```
P->496
del 8128
496 -> •
T->6
1->28

del 6

6)
```
for (Q=P; P != NULL; P=Q)
    Q = Q->getNext();
    delete P;
}
Head = P;
```
{del 1
 del 6
 del 28
 del 496
 del 8128

7)
```
LinkNode     *R=Head->get
Head = P->getNext();
delete P;
P = NULL;
R = R->getNext()->getNext
Q-> setNext(R->getNext())
delete R;
R = NULL;
Q = Q->getNext();
```

```
R->6
Head->6
del 1
P-> •
R->496
28->8128
del 496
R-> •
```

8)
```
opp( Head, Q->getNext()->getNext() );
// . . .
void opp( LinkNode* x, Lin
{
  LinkNode*  t;
  if (x != NULL && y != NU
    for (t=x; (t!=NULL &&
                t->getNext(
       t = t->getNext();
    int s = x->getData().g
    x->setData(y->getData(
    y->setData(s);
    opp(x->getNext(), t);
    }//if
}//opp
```

```
x->1
y->8128

t->Next->y
swap x's &
y's  values

swap next 2
```

9)
```
delete [4] Head->getNext();
Head->setNext(NULL);
```

```
ERROR:
not an
array
pointer
```

10) None of the above

III. Separate Compilation

For the next two questions, consider a C++ program composed of three `cpp` files and three corresponding header files, as shown below. All function calls are shown, as are all `include` directives, type declarations and function prototypes. 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.

```
// main.h
//. . .
class MainClass {
   //. . .
};
```

```
// main.cpp
#include "main.h"
#include "ClassToo.h"
#include "Test2.h"
//. . .
void main() {
   ClassToo C;
   Test2(C);
   //. . .
}
```

9: When main.cpp is compiled the inclusion of ClassToo.h results in a second copy of main.h being included and thus the compiler encounters a second definition for MainClass.

```
// Test2.h
//. . .
void Test2(ClassToo C2obj);
//. . .
```

```
// Test2.cpp
#include "Test2.h"
//. . .
void Test2(ClassToo C2obj) {
   //. . .
}
```

10: When Test2.cpp is compiled, the compiler sees that the Test2() function accepts a ClassToo parameter. Since ClassToo.h has not been included the definition is missing from the Test2.cpp scope.

```
// ClassToo.h
#include "main.h"
//. . .
class ClassToo {
  private:
      MainClass Mo
  //. . .
};
```

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

9. If the organization shown above is used, and no preprocessor directives are added, what will the compiler complain about when `main.cpp` is compiled?

1) Nothing.
2) Multiple definitions for `MainClass`.
3) Multiple definitions for `ClassToo`.
4) Multiple definitions for `Test2()`.
5) Both 2, 3 and 4.
6) Undeclared identifier `MainClass`.
7) Undeclared identifier `ClassToo`.
8) Undeclared identifier `Test2`.
9) None of these.

10. If the organization shown above is used, and no preprocessor directives are added, what will the compiler complain about when `Test2.cpp` is compiled?

1) Nothing.
2) Multiple definitions for `ClassToo`.
3) Multiple definitions for `Test2()`.
4) Both 2 and 3.
5) Undeclared identifier `ClassToo`.
6) Undeclared identifier `Test2()`.
7) Both 5 and 6.
8) 2, 3, 5, and 6
9) None of these.

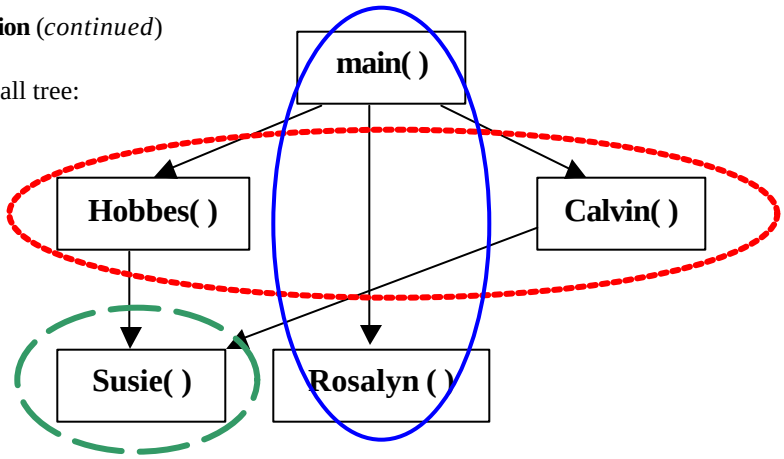
11. (True or False) The order of the `#include` statements in `main.cpp` does not matter, (i.e. if `#include "Test2.h"` is listed above `#include "main.h"` the same compilation as above would result?

1) True

11: If Test2.h were placed above main.h the compiler would also complain about an undecalared identifier ClassToo in the Test2() prototype since the prototype declaration would now precede the ClassToo declaration in main.cpp.

III. Separate Compilation (*continued*)

Consider the function call tree:



Assume that the software system is to be decomposed for compilation into three separate source files: `main.cpp`, `Calvin.cpp`, and `Susie.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`
`Rosalyn( )`	`main.cpp`
`Calvin( )`	`Calvin.cpp`
`Hobbes( )`	`Calvin.cpp`
`Susie( )`	`Susie.cpp`

`Scott separate compilation unit`

```
//Calvin.h
void Calvin ( /* parameters */ );
```

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

void Calvin ( /* parameters */ ){
// Calvin's code
   Susie();
}

void Hobbes ( /* parameters */ ){
// Hobbes's code
   Susie();
```

`Susie separate compilation unit`

```
//Susie.h
void Susie ( /* parameters */ );
```

```
// Susie.cpp
#include "Susie.h"

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

}
```

`main separate compilation unit`

```
//main.h
/* main declarations */
```

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

void main() {

   Hobbes  ( /* parameters */ );
   Rosalyn ( /* parameters */ );
   Calvin  ( /* parameters */ );

}

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

}
```

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** "`Calvin.cpp`" is **compiled** (not built) within Microsoft Visual C++, which of the following type of errors would occur:

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

12: When Calvin.cpp is compiled the call to Susie(), [shown on the call tree], requires that the prototype for Susie be in the same scope.

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

```
1) void Rosalyn ( /* parameters */ );     3) void Calvin ( /* parameters */ );

2) void Hobbes( /* parameters */ );       4) void Susie ( /* parameters */ );
```

Since Hobbes() is called by main() which is in another .cpp file.

14. In addition to the include directives listed above, where else **should** "`Susie.h`" be included?

```
(1) main.h        (3) Calvin.h = 0.5 (5) Susie.h
(2) main.cpp      (4) Calvin.cpp     (6) nowhere else
```

14: Susie() is called by Calvin() so her prototype must be in the same scope as Calvin.

15. In addition to the include directives listed above, where else **should** "`Calvin.h`" be included?

```
(1) main.h = 0.5  (3) Susie.h       (5) Calvin.h
(2) main.cpp      (4) Susie.cpp     (6) nowhere else
```

15: Calvin() is called by main() so his prototype must be in the same scope as main.

16. Assume the partial code above was completed and contained no compilation or linking errors and that the files were contained within a MSVC project. How many different object files (`.obj`) would the MSVC project build produce?

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

16: An object file is only produced for each separate .cpp file.

IV. Object Manipulations

Assume the following class declaration and implementation:

```
class Watterson {
private:
   char* comic;
public:
   Watterson (char com='*');
   char getComic() const;
   void setComic(char com);
   bool operator==(
        const Watterson& watt) const;
   ~Watterson();
};

Watterson::Watterson (char com) {
   comic = new char(com);
}
```

```
char Watterson::getComic() const {
   return(*comic);
}

void Watterson::setComic(char com) {
   *comic = com;
}

bool Watterson::operator==
     (const Watterson& watt) const {
   return ( int(*comic) ==
            int(*(watt.comic)) );
}

Watterson::~Watterson () {
    delete comic;
}
```

Given the following code:

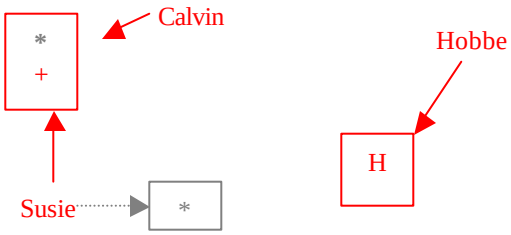
```
char Wormwood (Watterson Moe);

void main() {
   Watterson Calvin, Hobbes('H');
   Watterson Susie;

   Susie = Calvin;
   Susie.setComic('+');

   cout << "Contents of Calvin is:" << Calvin.getComic() << endl;    //LINE 1
   cout << "Contents of Susie is:"  << Susie.getComic()  << endl;    //LINE 2
   cout << "Contents of Hobbes is:" << Wormwood(Hobbes)  << endl;    //LINE 3
   cout << "Contents of Hobbes is:" << Hobbes.getComic() << endl;    //LINE 4
}

char Wormwood (Watterson Moe) { return ( Moe.getComic() ); }
```



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

1) '*'
2) '+'
3) 'H'
4) Execution Error
5)

17. What character is output by the call `Calvin.getComic()` in `LINE 1` above? **2**

18. What character is output by the call `Susie.getComic()` in `LINE 2` above? **2**

19. What character is output by the call `Wormwood(Hobbes)` in `LINE 3` above? **3**

20. What character is output by the call `Hobbes.getComic()` in `LINE 4` above? **4**

Default construction of Calvin & Susie allocates an '*' char. Construction of Hobbes allocates an 'H' char. Susie assigned to Calvin results in a member-wise shallow copy (& a memory leak) and pointer aliases. Setting Susie's comic to a plus results in Calvin's also being set due to the pointer alias.

21. (True/False) The above code contains a memory leak.

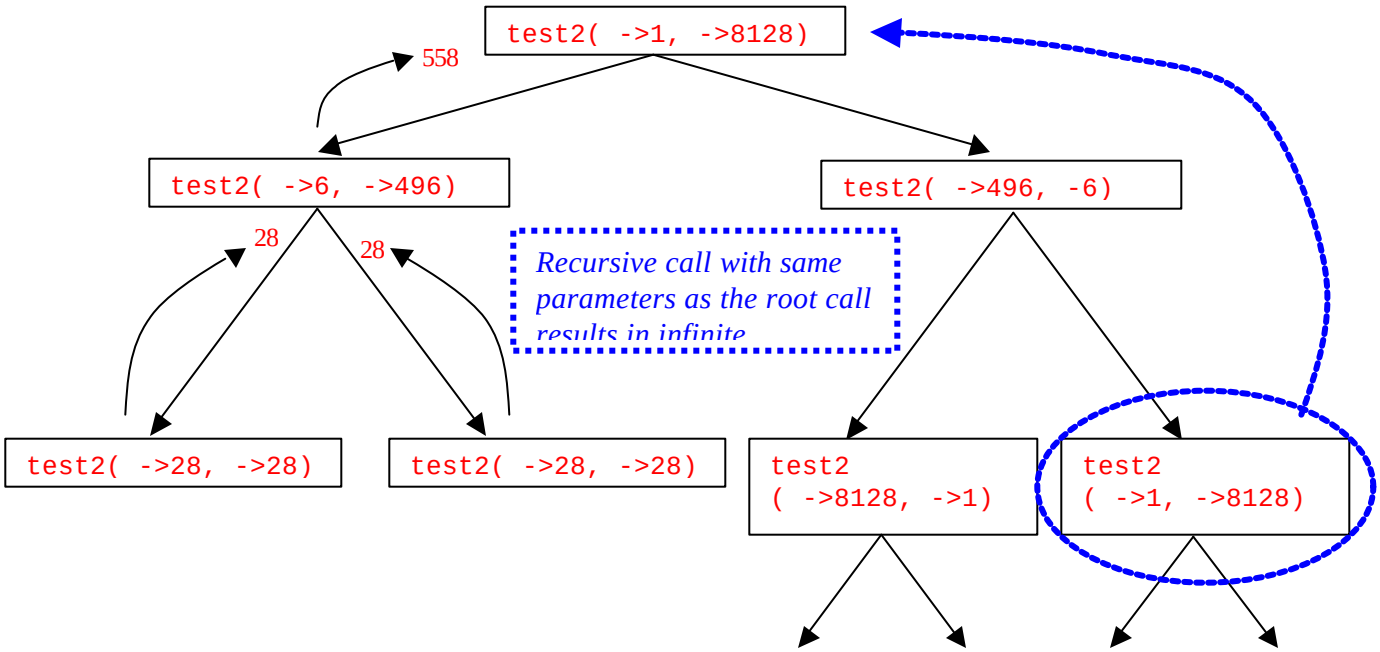
1) True 2) False

V. Recursion

22. The execution of a recursive function results in activation records for the function being created and stored where?

1) Heap
2) Runtime Stack
3) Registers
4) Text Segment
5) Data Segment
6) None of the above

23. Recursive tree trace



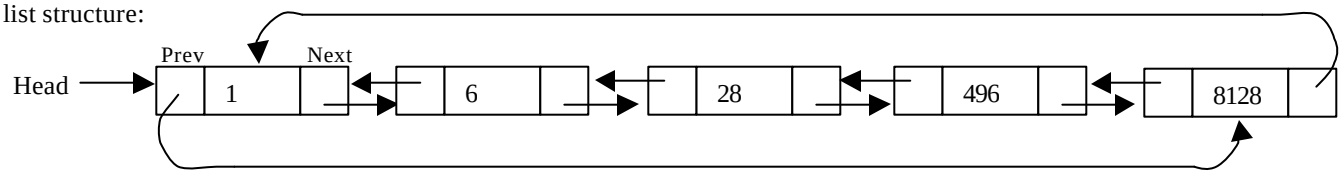
Consider the linked list class and list node declarations given below:

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

```
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;
```

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



23. What is the value returned by the call `cout << test2(Head, Head->getPrev());` to the following recursive function:

```
int test2 (LinkNode *L, LinkNode *R) {

if ( (L == NULL) || (R == NULL) ) return 0;
else if (L->getData().getValue() == R->getData().getValue())
         return (R->getData().getValue());
else {
     int left  = L->getData().getValue();
     int right = R->getData().getValue();
     int Lsum = test2(L->getNext(), R->getPrev());
     int Rsum = test2(R->getPrev(), L->getNext());
     return( left + Lsum + Rsum + right );
   }
}
```

1) 28
2) 531
3) 8128
4) 8659
5) 17318
6) 25977
7) 34636
8) 43295
9) 51954
10) None of the above
(prev page has trace)

VI. Recursion *(continued)*

For the next two questions, consider the following recursive function:

```
int Lie( int i )
{
     if (i < 9)
          return( 13 ) ;
     else if (i < 10)
          return( 21 ) ;
     else
          return( Lie(i-2) + Lie(i-1) ) ;

} // Lie
```

24. What is returned from the call: `Lie (15)` ? (*Hint: don't do the same work twice*.) *(next page has trace)*

1) 13
2) 21
3) 34
4) 55
5) 89
6) 144
7) 233
8) 377
9) 610
10) None of the above

25. Not counting the original call, how many recursive calls are made by the execution of: `Lie (15)` ?

1) 1
2) 2
3) 3
4) 5
5) 8
6) 13
7) 21
8) 34
9) 55
10) None of the above

24/25 Recursive tree trace