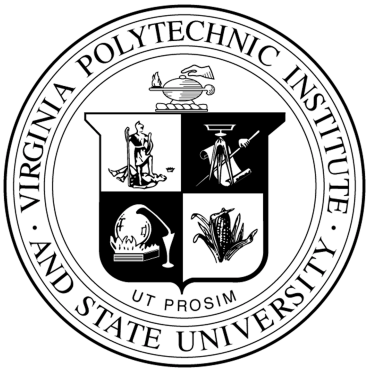


READ THIS NOW!

- 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**'.
- Code Form **A** on your Opscan. Check your SNN 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 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 4 questions, assume the following declarations:

```
class   Entity;
typedef Entity*  Bond;
class   Entity {
private:
   Bond      connect;
   int       item;
public: //member functions
        void snaf();
};
```

```
//inside the Entity member function snaf
Entity part;
part.item = 10;
part.connect = NULL;
connect = &part;
*this = part;
item = 20;
connect = new Entity;
connect->item = 30;
connect->connect = this;
```

1. From inside the same member function as the above code, what is the data type of the expression at the right:

```
this->connect
```

1) NULL
2) Entity
3) Entity*
4) item*
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 link, (i.e., point), the `Entity` containing the integer `10` to the `Entity` containing the integer `30`?

1) `part->connect = this;`
2) `part.connect = connect;`
3) `connect->connect = part.connect;`
4) `this->connect = &part;`
5) `*this.connect = new Entity(30);`
6) None of the above

3. From inside the same member function as the above code, what would be the data type of the expression at the right?

```
*(connect->connect)
```

1) NULL
2) Entity
3) Entity*
4) item*
5) Bond*
6) None of the above

4. 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 `Entity` objects would still exist in memory?

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

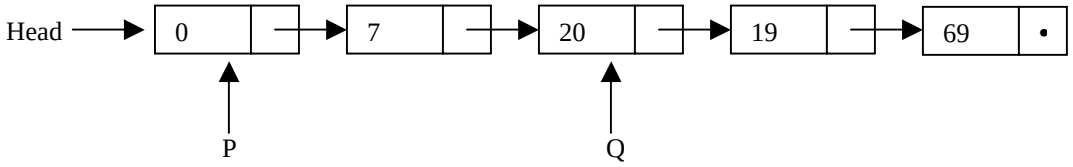
II. Linked List Class Manipulation

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;}
};
```

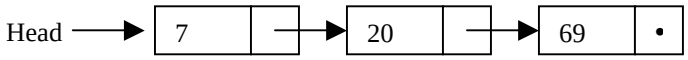
```
class LinkNode {
private:
   ItemType  Data;
   LinkNode* Next;
public:
   LinkNode();
   LinkNode(ItemType newData);
   bool setNext(LinkNode* newNext);
   bool setData(ItemType  newData);
   ItemType  getData() const;
   LinkNode* getNext() const;
};

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:

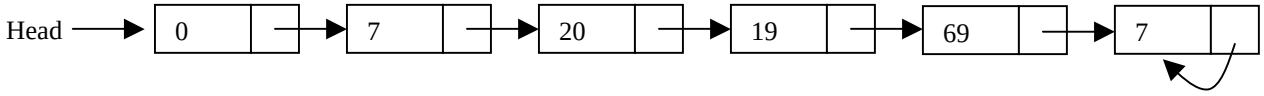


For the next 4 questions, select from the code segments on the following page, the segment that would transmogrify the above list into each of the lists shown below. Assume the list structure above as your starting point (for each question). Choose from the possible answers given on the following page.

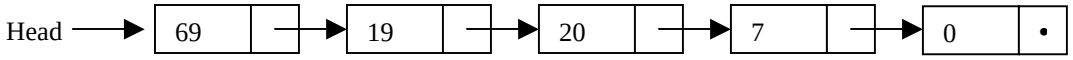
5.



6.



7.



8.

II. Linked List Class Manipulation (*continued*)

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

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

2)
```
LinkNode     *S=Head;
Head = NULL;
delete S;
```

3)
```
LinkNode     *T=Head;
while (T->getNext()->getNext()
                          != NULL) {
   T = T->getNext();
}
T->setNext(new LinkNode);
T = T->getNext();
T->setNext(T);
T->setData(7);
```

4)
```
LinkNode     *T=Head;
while (Q->getNext()!= NULL)
  Q = Q->getNext();
Head = Q;
for (int i=0; i<4; i++) {
  P = T;
  while (P->getNext()!= Q)
        P = P->getNext();
  Q->setNext(P);
  Q = P;
}
T->setNext(NULL);
```

5)
```
for (int i=0; i<2; i++)
    Q = Q->getNext();
LinkNode *T = new LinkNode;
Q->setNext(T);
T->setNext(Q->getNext());
T->setData(7);
```

6)
```
for (Q=P; P != NULL; P=Q) {
   Q = Q->getNext();
   delete P;
}
Head = P;
```

7)
```
LinkNode     *T=Head;
while (T->getNext()->getNext()
                          != NULL) {
   T = T->getNext();
}
Q->setNext(T->getNext());
delete T;
Head = P->getNext();
delete P;
```

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

9)
```
delete [5] Head;
Head = NULL;
```

10) None of the above

III. Separate Compilation

For the next two questions, consider a C++ program composed of two `cpp` files and two 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.

```
// classes.h
. . .
class Thing {
   . . .
};
```

```
// main.cpp
#include "classes.h"
#include "fun.h"
. . .
int main() {
   Thing T;
   Fn(T);
   . . .
}
```

```
// fun.h
. . .
void Fn(Thing obj);
```

```
// fun.cpp
#include "fun.h"
. . .
void Fn(Thing obj) {
   . . .
}
```

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 `Thing`.
   3) Multiple definitions for `Fn()`.
   4) Both 2 and 3.
   5) Undeclared identifier `Thing`.
   6) Undeclared identifier `Fn()`.
   7) Both 5 and 6.
   8) 2, 3, 5, and 6
   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 `fun.cpp` is compiled?

    1) Nothing.
    2) Multiple definitions for `Thing`.
    3) Multiple definitions for `Fn()`.
    4) Both 2 and 3.
    5) Undeclared identifier `Thing`.
    6) Undeclared identifier `Fn()`.
    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 "fun.h"` is listed above `#include "classes.h"` the same compilation as above would result?

    1) True
    2) False

IV. Object Manipulations

Assume the following class declaration and implementation:

```
class Arness {
private:
   bool* thing;
public:
   Arness(bool smoke=true);
   bool getThing() const;
   void ThingT();
   void ThingF();
   bool operator!=(
               const Arness& festus);
   ~Arness();
};

Arness::Arness(bool smoke) {
   thing = new bool(smoke);
}

bool Arness::getThing() const {
   return(*thing);
}
```

```
void Arness::ThingT() {
   *thing = true ;
}

void Arness::ThingF() {
   *thing = false ;
}

bool Arness::operator!=
           (const Arness& festus) {
   return (*thing !=
           festus.getThing());
}

Arness::~Arness () {
    delete thing;
    thing = NULL;
}
```

Given the following code:

```
string Alien(const Arness& doc);

void main() {
   Arness New, MissK(true);
   Arness Sam = MissK;

   Sam.ThingF();

   cout << "State of MissK is:" << Alien(MissK) << endl;       //LINE 1
   cout << "State of Sam is:" << Alien(Sam) << endl;           //LINE 2
   Sam.~Arness(); //Sam destroys himself
   cout << "State of MissK is:" << Alien(MissK) << endl;       //LINE 3
}

string Alien(const Arness& doc) {
    if (doc.getThing()) return “true”;
    else return “false”;
}
```

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

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

12. What is output by the call `Alien(MissK)` in `LINE 1` above?

13. What is output by the call `Alien(Sam)` in `LINE 2` above?

14. What is output by the call `Alien(MissK)` in `LINE 3` above?

IV. Object Manipulations *(continued)*

15. Consider a class that contains and allocates dynamic memory. If that class's member functions contain no *deep copy* logic code then logic errors may result in all 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 assigned to an existing object of the same class.

    4) When an object is passed by value.

    5) None of the above, (all of the above situations would result in a *shallow* copy).

16. Consider the `LinkNode` and `LinkList` classes discussed in class. Which of the classes should have a destructor function implemented?

    1) LinkNode 2) LinkList 3) Both 1 & 2 4) Neither 1 or 2

V. Command Line Arguments

Consider the P3 LCIS program that provides for possibly one command line argument:

```
LCIS <InitialCISAreaDataFileName>
```

Given the input file stream definition:

```
ifstream CISiFile;
```

17. Which of the following will correctly open the file stream with the command line argument:

1) `CISiFile.open(argc[0]);`
2) `CISiFile.open(argc[1]);`
3) `CISiFile.open(argv[0]);`
4) `CISiFile.open(argv[1]);`
5) `CISiFile.open(argc[1].c_str());`
6) `CISiFile.open(argv[1].c_str());`
7) `CISiFile.open(argc+1);`
8) `CISiFile.open(argv+1);`
9) `CISiFile.open(*(argv+1));`
10) None of the above

VI. Recursion

18. If a recursive function is illogically coded, such that the base case code never executes, which of the following errors would most likely result?

1) Heap Exhaustion
2) Runtime Stack Overflow
3) Register Depletion
4) Memory Leak
5) Virtual Alias Pointer
6) None of the above

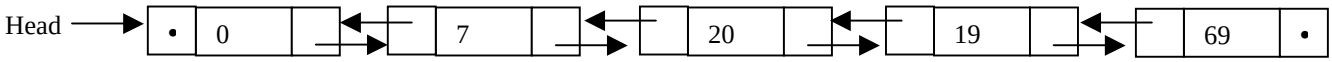
VI. Recursion *(continued)*

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:



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

```
int test2 (LinkNode *M) {

if (M == NULL) return 0;
else if (M->getData().getValue() == 7)
         return (M->getData().getValue()*1000000);
     else if (M->getData().getValue() > 50)
         return (M->getData().getValue());
     else if (M->getData().getValue() < 20)
         return ( M->getData().getValue()*100 +
                  test2(M->getNext()) );
     else return (M->getData().getValue()*10000 +
                  test2(M->getPrev()) +
                  test2(M->getNext()) );
}
```

1) 0
2) 7
3) 19
4) 20
5) 69
6) 1900
7) 7000000
8) 200000
9) 7201969
10) None of the above

VI. Recursion *(continued)*

20. Consider the following recursive function:

```
int ReCurse( int k )
{
     if (k == 1)
           return( 1 ) ;
     else
           if ( (k % 2) != 0 )
                 return( 2 + ReCurse ( 3*k+1 )  ) ;
           else
                 return( 1 + ReCurse ( k/2 ) );

} // ReCurse
```

What is returned from the call: `ReCurse (6)` ?

1) `11`
2) `2`
3) `3`
4) `4`
5) `5`
6) `None of these`
7) `7`
8) `8`
9) `9`
10) `10`