Problem #1

(10 points) The program below has a series of print statements. Each one has a comment line after it with a letter in parenttheses like lis :// (A)
Next to each letter, write the output that you expect to see from that statement when the program runs. All these statements are part of one program, so you should answer the questions in order.

```
int a[ ] = {5, 6, 7, 8};
int &b = a[0];
int &c = b;
c = 0;
cout << a[0] << "," << b << "," << c << endl;

\\ (A) ________________________________________

int d = a[1];
int * e = & a[2];
d = 0;
* e = 0;
cout << a[1] << "," << a[2] << "," << d << "," << *e << endl;

\\ (B) ________________________________________

int * f = & a[3];
f = e;
*f = 20;
cout << a[2] << "," << a[3] << "," << *e << "," << *f << endl;

\\ (C) ________________________________________
```

Problem #2

(10 points) Here is a for statement from a recent piece of CS61B code:

```
int prod = 1;
for (int i = 1; i<=n; i++) {
prod *= i;
// "Loop invariant"
}
```

Which of the following would be a suitable loop invariant to replace the " Loop invariant " in the comment above?

(a) prod *= i;

(b) prod = i! (i factorial)
(c) i = i + 1
(d) i++

Your answer ____________________ (4 points)

Now consider the following piece of code:

```
int sum = 1;
for (int i = 0; i < n; i++) {
sum += sum;
// "Loop invariant"
}
```

write down a loop invariant to replace the "Loop invariant" comment:

Your answer __________________________ (6 points)

Problem #3

(10 points) Write a function named "remvowel" that removes vowels from a string argument (producing C++-like names). Your function should not return a value, but should modify the string passed to it so that there are no vowels (none of 'a', 'e', 'i', 'o', 'u'). You should structure your solution carefully. Write out your specification, design and implementation (code) in the space below. You can use extra sheets (starting with the back of this page) if you need to. For design, just include a few sentences explaining how you plan to solve the problem (e.g. recursion, iteration etc.)

Specification: (2 points)

Design: (3 points)

Implementation: (5 points)

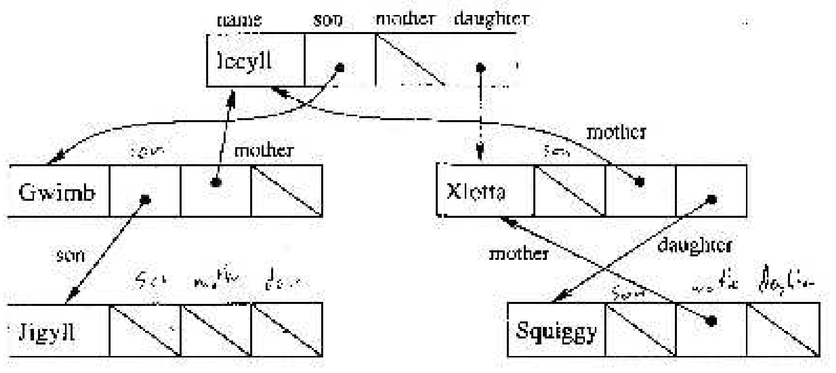
Problem #4

(10 points) The Gluxx are a highly intelligent, gelatinous race of being about 5 years warp drive from earth. On a recent visit, the naturalist C. Darwin (no relation) discovered that Gluxx families have a mother and a father, and at most one son and one daughter. He discovered that Gluxx always draw their family trees with the motehr only (**footnote). He built several family trees using the data structure below:

```
struct gluxx {
char * name;
gluxx * mother;
gluxx * son;
gluxx * daughter;
};
```

"Damn those C structs!" he exlaimed on return to earth, realizing that he had forgotten to include a gender field (M or F for Gluxx), and wishing to blame the programming language for his own forgetfulness. Fortunately, he realized that he could often figure out the gender from the data. Below is a typical family tree. Note that every Gluxx still has a father, but there is no pointer for fathers. This tree satisfies the following rule:

RULE: If Gluxx A and B occur in the tree, and B is the (son, mother, daughter) of A, then there must be a (son, mother, daughter) pointer from A to B.

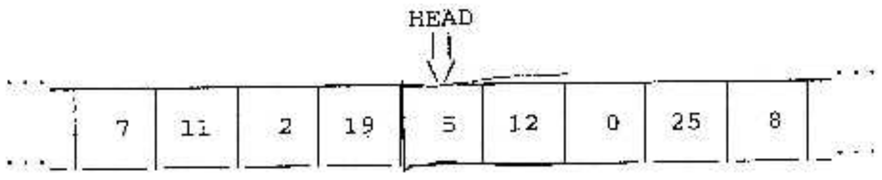


**footnote: Just like humans, Gluxx get all their Mitochondrial DNA from their mother. While not important to humans, bad Gluxx M-DNA can lead to health problems like curdling and crusting.

Write a function char * GluxxGender(gluxx *) that when given a pointer to a single Gluxx node, tries to figure out the gender of the Gluxx it points to. The function should return string "M", "F", or "No Idea". Include specification, design and implementation. Clearly label those parts of your answer with "Specification", "Design" and "Implementation".

Problem #5

(20 points) Acme Turing Mcahine Company is hiring. In your interview with them, you are asked to implement a C++ abstract data type called a "tape". A tape looks like this:



Initially, the tape is filled with zeros. The tape is assumed to be indefinitely long in both directions, but you don't need to be concerned about that because of the implementation you will use.

Here are the operations to be supported by the tape type:

int MoveLeft (); // Move the head one square left and return the new number under it.

```
int MoveRight (); // Move the head one square right and return the new number under it.
void WriteVal (int); // Write an integer argument on the tape square under the head.
```

You don't have enough time to write the code for the tape from scratch, but luckily, on the back of your business cards, you have the code for the CS61B implementation of stacks. You realize in a stroke of inspiration that a tape is just two stacks placed top-to=top. That is, one stack holds the data to the left of the head and the other holds the data to the right of the head (and the square under the head). Moving the tape head is the same as popping a value from one stack and pushing it onto the other. Here is the public part of the declaration of the stack type:

```
class stack
{
public:
void push(int, val);
int pop();
int stack_empty(): // returns 1 if stack empty, 0 otherwise.
stack(int InitialSize);
};
```

(a) (5 points) Write the class declaratin for a tape, including two stacks as private member fields:

(b) (5 points) Write the constructor for tapes. Be careful how you call the stack constructor!

(c) (5 points) Now write the WriteVal member function.

(d) (5 points) Write the MoveLeft function. Don't forget about empty stacks!! Because the tape is supposed to be infinite, it is legal to move to a square that hasn't been visited before. One stack will be empty then, but that is OK. Empty data cells are assumed to contain zero.

Posted by HKN (Electrical Engineering and Computer Science Honor Society)
University of California at Berkeley
If you have any questions about these online exams
please contact examfile@hkn.eecs.berkeley.edu.