1. **[35 points in all]**

Consider the following code, which may or may not have bugs or use bad style:

```
class Hillary
{
  public:
    Hillary()  { cout << "H "; }
    ~Hillary() { cout << "~H "; }
};

class Bernie
{
  public:
    Bernie(int i)              { cout << "B "; m_id = i; }
    virtual ~Bernie()          { cout << "~B "; }
    int id() const             { return m_id; }
    void marco() const         { cout << m_id << " "; }
    virtual void ted() const   { cout << (m_id+2) << " "; }
    virtual void incr()        { m_id++; }
  private:                                              m_id - k = 0.
    int     m_id;-k.
    Hillary m_john[2];
};            H H . B        ~B.~H ~H

class Donald : public Bernie
{                                cout << " D "
  public:
    Donald(int i);             // You implement this in part a
    ~Donald()                  { cout << "~D "; }
    void marco() const         { cout << (id()+3) << " "; }
    virtual void ted() const   { cout << (id()+4) << " "; }
  private:
    Hillary m_ben;
};

void process(Bernie* bp)
{
    bp->ted();      // line e
    bp->marco();    // line f
    delete bp;      // line g
}

int main()
{
    int yourID;        // You enter the last digit of your
    cin >> yourID;     // student ID.
                               3
    process(new Bernie(yourID)); cout << endl;    // line b
    process(new Donald(yourID)); cout << endl;    // line c
                               3
}
```

(Questions are on the next page)

H-H-B_5_3~B~H~H

H H B $\overset{\wedge}{}$ D_7_6~D~H~B~H~H
H

a. Making no additions or changes to any of the code on the previous page, implement here the Donald constructor so that the code segment

```
int k = ...some value...;
Donald d(k);
cout << (d.id() - k);
```

=0.

always writes output that ends with `D 0` (i.e., the last three characters written are capital D, space, and zero. Note that the last line above must be what writes the 0.

```
Donald::Donald(int i) : Bernie(i)
{ cout << "D ";
}
```

b. Assuming you've correctly implemented the constructor of part a, if you enter the last digit of your student ID number when the main routine is executed, what is the line of output produced by the code in line b of the main routine?

"H H B 5 3~B~H~H"

c. Under the same conditions as part b, what is the line of output produced by the code in line c of the main routine?

"H H B H D 7 ~~6~D~H~B~H~H"~~

d. Suppose we added `virtual void incr();` to the public part of the Donald class. Assuming you've correctly implemented the constructor of part a, and making no additions or changes to any of the code above, implement `Donald::incr` so that it increments the object's ID number by 1 if the ID number is greater than 6, and leaves it unchanged if it is not.

```
void Donald::incr()
{
    if (id() > 6)
        Bernie::incr();
}
```

e. Of the lines labelled b, c, e, f, and g in the `process` and `main` functions, which one(s), if any, would produce a compilation error if the definition of `Bernie::ted` were changed to `virtual void ted() const = 0;`? Answer with just the line label(s) or *none*; we're not asking for an explanation.

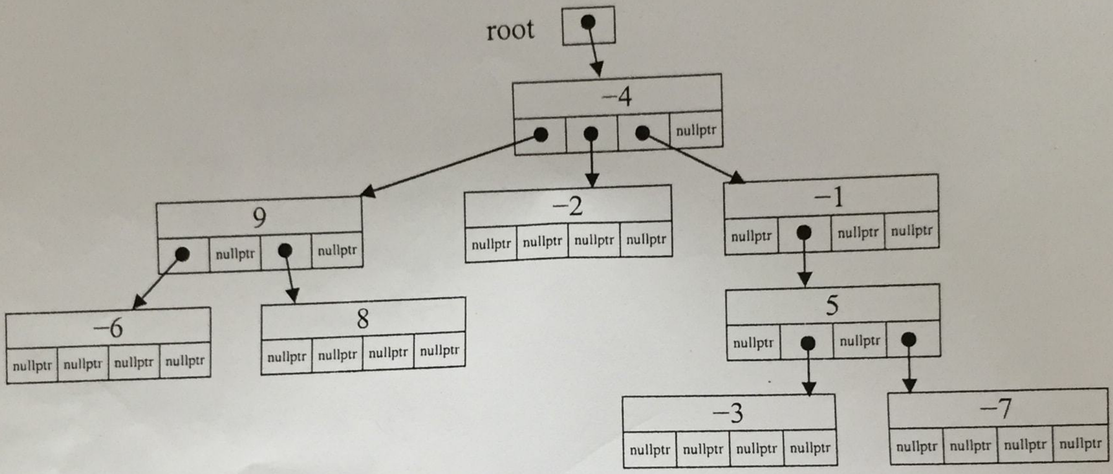
2 e b

2. **[15 points]**

A 4-tree is a data structure that employs nodes defined as follows:

```
struct Node
{
    int data;
    Node* child[4];
};
```

Each element of the `child` array is either nullptr, denoting an empty subtree, or a pointer to a non-empty subtree. Here's a picture of one possible 4-tree:


root
-4
nullptr
9
nullptr
nullptr
-2
nullptr nullptr nullptr nullptr
-1
nullptr
nullptr nullptr
-6
nullptr nullptr nullptr nullptr
8
nullptr nullptr nullptr nullptr
5
nullptr
nullptr
-3
nullptr nullptr nullptr nullptr
-7
nullptr nullptr nullptr nullptr

The variable `root` is a `Node*` that points to a node with the data −4. That node's four children are the tree rooted at the node whose data is 9, the one-node tree whose data is −2, the tree rooted at the node whose data is −1, and an empty tree.

Write a **recursive** function named `fixNegatives` that takes as its one parameter a pointer to a node and replaces every negative data value with 0 in the tree rooted at that node, returning the number of values so replaced (which will be 0 if the tree is empty). For example, the statement

```
int n = fixNegatives(root);
```

would replace the six negative values in the tree shown above with 0, and set `n` to 6. If instead of that statement, we had executed the statement

```
int n = fixNegatives(root->child[2]);
```

with the tree shown above, the three negative values −1, −3, and −7 would be replaced by 0, and `n` would be set to 3. The statement

```
int m = fixNegatives(nullptr);
```

would set `m` to 0, since the empty tree has no negative data values to replace.

(continued on next page)

Your function must use recursion in a useful manner. It must not declare any variables with the keyword `static`, and must not use any global variables. If you declare any local variables, they must be of type `int` or `Node*`. The body of your function must contain no more than 15 statements (this is more than enough). **Violating these constraints will result in a score of zero for this problem.** You may use a loop if you wish, as long as your function uses recursion in a useful manner.

Write your function here:

```
int fixNegatives(Node* p)
{
      if (p== nullptr)
          return 0;

      int count = 0
      if (p->data <0)
      {    p->data =0;
           count++;
      }
      for (int i=0; i<4 ;i++)
           count += fixNegatives(p->child[i]);

      return count;
 }
```

3. [15 points]

Here is a non-recursive implementation of Problem 2's `fixNegatives` function that in addition, writes the data at each node it visits. (For this code and any changes you make to it for part b, assume any required headers have been included and that `using namespace std;` is present.

toDo 9 -2 ~~-1~~

9 -2 ~~5~~

~~9~~ ~~-2~~ ~~-3~~ ~~-7~~

-6 ~~8~~

```
    int fixNegatives(Node* p)
    {
1       int count = 0;
2       queue<Node*> toDo;
3       toDo.push(p);
4       while ( ! toDo.empty())
        {                  top
5           p = toDo.front();
6           toDo.pop();
7           if (p != nullptr)
            {
8               cout << " " << p->data;  // write a number
9               if (p->data < 0)
                {
10                  p->data = 0;
11                  count++;
                }
12              for (int k = 0; k < 4; k++)
13                  toDo.push(p->child[k]);
            }
        }
14      return count;
    }
```

~~-4~~

~~9~~ -2 -1

~~-2~~ ~~-1~~ -6 8 5

~~-6~~ ~~8~~ 5

~~5~~ -3 -7

~~4~~

-1 -2 ~~9~~

-1 -2 8 -6

a. If this function is called with a pointer to the node whose data value is –4 in the tree shown in Problem 2, what is written by all the executions of the indicated output statement? Circle the letter of your choice, and if it's K, write the output.

A. -4
B. -4 -1 5 -7 -3 -2 9 8 -6
C. -4 -1 5 -7
D. -4 -1 -2 9
E. -4 -1 -2 9 5 8 -6 -7 -3
F. -4 -1 -2 9 5 -7 -3 8 -6
G. -4 9 -6 8 -2 -1 5 -3 -7
H. -4 9 -2 -1 5 -3 -7
I. -4 9 -2 -1 -6 8 -3 5 -7
J. -4 9 -1 5 -6 8 -2 -3 -7
[✓ K.] Something else. Write that something else here:

-4 9 -2 -1 -6 8 5 -3 -7

✓

4

4

9 -6

b. Indicate one-line replacements for no more than three of the numbered lines of the above function so that if it is called with a pointer to the node whose data is –4 in the Problem 2 tree, the output would be `-4 9 -6 8 -2 -1 5 -3 -7` (If no replacements are necessary, then write "No replacements".) Replacement text must not use the name `fixNegatives` (so don't introduce recursion into this code).

✓

Line number: 2 Replacement: stack<Node*> toDo;

Line number: 5 Replacement: p = toDo.top();

Line number: 12 Replacement: for (int k=3; k>=0; k--)

~~-4~~

9 -2 -1

9