

Key

You may consult one 8.5x11 inch sheet containing any information about the material covered in this course. This sheet must be turned in attached to the exam. No other materials, written or otherwise, are allowed other than the copy of the exam you are using and a pen or pencil. All questions about the exam should be addressed to the instructor or a teaching assistant. All questions in the exam should be answered on the exam or on the lined paper provided. Use only a blue or black pen or a dark (no. 2) pencil. (Note: The exam is divided into 130 points.)

1. (6 points) Write the output of the following code, as it would appear on the monitor screen. In this question, object `fileIn` is an `ifstream` object and has been properly opened to read from a file called “`data.dat`”.

```
File "data.dat":    4    3    2   -4    9   -8    1   -3

double num, total=0;

ifstream inFile >> num;
while (inFile && num > -10)
{
    if (num > 0)
        total = total - num;
    else
        total = total + num;
    inFile >> num;
}
cout << total;
```

-34

2. (12 points) Write the output of the following code, as it would appear on the monitor screen.

```
const int b = 0;

int DashRiprock(int h, int &i, int &j);

int main()
{
    int a=10, b=20, c=30, d=40;
    a = DashRiprock(c, d, b);
    cout << a << ' ' << b << ' ' << c << ' ' << d;

    return 0;
}

int DashRiprock(int h, int &i, int &j)
{
    h = h / 2;
    i = h;
    j = j / 4;
    return b;
}
```

0	5	30	15
---	---	----	----

Problems #6, #7 and #8 will depend on the following struct:

```
struct Purchases {
    string client;
    int hirudoMedicinalis;           // European Medicinal Leech
    int hirudinariaManillensis;     // Asian Medicinal Leech
    int haementeriaGuilianii;       // Amazon Leech
};
```

6. (10 points) Given the above struct, write a function that will receive one struct of type *Purchases*, a string name of a client and three quantities of numbers of leeches to be purchased. The function must assign the client value and the quantity of leeches to be purchased to the struct object. The struct object must be returned to the calling function either as a return or as a pass by reference. You do not have to write a prototype or a call.

```
void InsertValues(Purchases & p, string c, int worm1, int worm2, int worm3)
{
    p.client = c;
    p.hirudoMedicinalis = worm1;
    p.hirudinariaManillensis = worm2;
    p.haementeriaGuilianii = worm3;
}
```

7. (5 points) Given the above struct, write the code necessary to declare an array of size *MAX* of the struct *Purchases*.

```
Purchases Worms[MAX];
```

8. (15 points) Using the array of problem #7 and the function of problem #6, write the code that will read all of the data in the file "*leechpurchases.dat*" and place them in the array you created in problem #7. Write this code on a sheet of lined paper. Do not place more data in the array than it can hold. The file "*leechpurchases.dat*" has the following form:

Example of File "leechpurchases.dat"

```
Gotta Bruise
0
15
7
Cauliflower Eeer
4
1
1
:
:
Justa Boxer
6
6
6
end-of-file
```

Note: The first number is the first leech quantity as it is listed in the struct, the second number is the second leech quantity as it is listed in the struct, etc

Note: You do not have to write any code other than to declare the variables that you will need and the actual code that will do the work, i.e. a fragment is OK.

```
int inUse = 0, w1, w2, w3;
string c;
ifstream inFile;
inFile.open("leechpurchaes.txt");
getline(inFile, c);
inFile >> w1 >> w2 >> w3;
while (inFile && inUse < MAX){
    InsertValues(Worms[inUse], c, w1, w2, w3);
    inUse++;
    getline(inFile, c);
    getline(inFile, c);
    inFile >> w1 >> w2 >> w3;
}
```

The following partial class declaration will be used in problems #9 and #10.

```
class LaCucaracha
{
public:
    LaCucaracha();           // constructor that sets object to alive

    void StompBug();         // when called, has a 10% chance of killing the bug

    bool IsAlive();         // returns true if bug is alive, false if dead

    bool IsFood();          // returns true if there is food, false if not

    bool IsLight();         // returns true if there is light, false if not

    void Eat();              // eats the available food

    void ChangeDirection(); // changes the direction the bug is facing

    void RunLikeHeck();     // move the bug in the direction it is facing
    :
    :
};
```

9. (25 points) On a lined sheet of paper and using the *LaCucaracha* class definition, write the code that will declare an object of the class *LaCucaracha* and maneuver the infernal thing until it is dead.

Maneuvering a *LaCucaracha* object consists of the following:

- if the nasty thing is dead, halt
- if the nasty thing is alive and the light is on, the bug should be stomped at
- if the nasty thing is still alive after being stomped at, it should move
- if the nasty thing is alive and the light is off, it should eat if there is food
- if the nasty thing is alive and the light is off, but there is no food, it should change direction and move

```
LaCucaracha roach;
while (roach.IsAlive())
{
    if (roach.IsLight())
    {
        roach.StompBug();
        if (roach.IsAlive())
            roach.RunLikeHeck();
    }
    else
        if (roach.IsFood())
            roach.Eat();
        else
        {
            roach.ChangeDirection();
            roach.RunLikeHeck();
        }
}
```

10. (35 points) Researchers at Lumbering State U. (obviously located in Oregon) are trying to breed a superior cockroach. Out of 100 female cockroaches, they want to find out which cockroach's eggs should be allowed to hatch. Since their supply of breeding cockroaches is too valuable to risk, they want to test their ideas by running a simulation where 100 simulated cockroaches will be forced to run until they expire. The cockroaches will not be provided with food or water, so they will not be allowed to eat during the simulation. However, each object in the array be continuously subjected to light and will be repeatedly stomped at during each pass through the logic of the simulation. If a cockroach survives a stomping, it will have to change direction and move. Thus any vermin left alive will have to continually be on the move.

On a sheet of lined paper, create a an array of 100 objects of class *LaCucaracha*, then write code to implement the above described simulation. With each pass through the logic of the simulation, each object in the array will be stomped at and, if it survives the stomping, will change directions and move. With each pass through the logic of the simulation, the index of any object in the array that is alive after the stomping will be output to the monitor screen. The simulation will halt when all objects in the array are dead.

Example 1:

```

LaCucaracha roaches[100];
bool allDead = false;
while (!allDead) {
    for (int x=0; x<100; x++)
        if (roaches[x].IsAlive())
            roaches[x].StompBug();
    for (int x=0; x<100; x++)
        if (roaches[x].IsAlive()){
            roaches[x].ChangeDirection();
            roaches[x].RunLikeHeck();
            cout << x << ' ';
        }
    cout << endl;
    allDead = true;
    for (int x=0; x<100; x++)
        if (roaches[x].IsAlive())
            allDead = false;
}

```

Example 2:

```

LaCucaracha roaches[100];
bool allDead = false;
while (!allDead)
{
    for (int x=0; x<100; x++)
    {
        allDead = true;
        if (roaches[x].IsAlive())
        {
            roaches[x].StompBug();
            if (roaches[x].IsAlive())
            {
                roaches[x].ChangeDirection();
                roaches[x].RunLikeHeck();
                allDead = false;
                cout << x << ' ';
            }
        }
    }
    cout << endl;
}
}

```