Project 1. Swiss Cheese

Part A

The compiler implements alignment rules on objects in response to hardware restrictions. For example, on most architectures, a 4-byte int should (or must) be stored at an address that is divisible by 4. The compiler ensures that class instances are allocated at addresses that are suitably aligned, and in laying out class data members, the compiler locates each one at a suitable address boundary. Since it never alters the order from the order in which the programmer listed them, it inserts unnamed holes where necessary to conform to alignment restrictions. This can be detected in the overall size of the resulting objects. Consider the following example:

```cpp
#include <iostream>
class C {
    char c;
    int i;
};
int main() {
    std::cout << sizeof(C) << std::endl;
}
```

This program prints the value 8, thus disclosing the existence of a 3-byte hole between the two data members of the class.

The holes between members that are primitive types are predictable from the hardware restrictions of the given architecture. Less predictable is the fact that the compiler may insert padding after the final data member. And when data members are themselves classes rather than primitive objects, it is not so easy to predict what kinds of alignment restrictions the compiler will enforce.

Here is a theory that we will investigate in this project. Our compiler (g++ version 3.1.1 or later) seems to associate with each class a quantity we shall call a “unitsize.” It then aligns any instance of a class according to the class’s unitsize. It also pads the class at the end so that its overall size is a multiple of its unitsize.

Do some experiments to confirm this theory.

Describe how the unitsize is determined. That is, give a rule that can be used to predict the unitsize in general.

(over)
Part B

Let us say a class is “admissible” if and only if every data member is: (1) of type char; (2) of type int; or (3) an admissible class. For example, the following is an admissible class:

```cpp
class {
    int x;
    class {
        char a,b;
    } ab;
    char c;
};
```

Define the “net size” of an admissible class as the sum of the net sizes of all its members, where the net size of a member of type char is `sizeof(char)` and the net size of a member of type int is `sizeof(int)`. The net size simply measures the total number of bytes contained directly or indirectly in char or int data members.

Define the “gross size” of an admissible class as the value returned by applying the `sizeof` operator to it.

Now define the “efficiency” of an admissible class as the ratio of its net size to its gross size. For example, the class C above has an efficiency of 0.625.

Your task is then to create an admissible class with the worst efficiency possible. This “swiss cheese” design will produce objects with lots of holes! It will be your holy grail in this project.

There may be no single best solution to this problem. (This confirms its standing as a kind of holy grail.) That is, there may be a sequence of examples of increasing size producing decreasing efficiency. In order to avoid the difficulties inherent in producing and grading infinitely large examples, you can stop work if you find an example whose efficiency is less than 1/3.

Show your best example as a set of declarations that produce an admissible class with the claimed efficiency.

Format for results

Your results should be in a file called `project1.cc`. The answer to Part A should be written in a concise comment at the top of the file. Following that should be the declarations creating your solution to Part B. Following that should be a `main()` that applies `sizeof()` to your class, that does a computation that calculates the net size of the class, and then a printout showing both those integer values as well as their ratio (net size / gross size) as a floating value. During grading, the program will be compiled and run in order to demonstrate your results. The program source code will also be scrutinized, with an amount of scrutiny proportional to how low the claimed efficiency is.

Submit your file by Tuesday, Feb. 24 at 5:00 p.m. using this command:

```
provide 150C++ project1 project1.cc
```