CSCI-1200 Data Structures — Spring 2016
Test 1 — Solutions

1 Sparse Matrix [/25]

A sparse matrix is a matrix in which most of the elements are zero. In this problem you will implement a sparse matrix using a simple class named `Sparse`. The sparse matrix class will store data in an STL vector, where each element of the vector represents a row of the matrix. Each element of the vector will be an STL list, representing columns. That is, the data will be stored in an STL vector of STL lists.

Initially, each element in the vector of rows will be an empty list. As data is added to the matrix, items are added to the list of the appropriate row. The list for each row will only contain elements with non-zero entries. The lists are maintained in order of increasing column index. For example, if row 1 contains non-zero elements at (1,1) and (1, 3). The list for row 1 will contain an element for column 1, followed by the element for column 3.

The elements in the column lists are Node objects which contain the column number and the data at that row and column position. Here is the Node class definition.

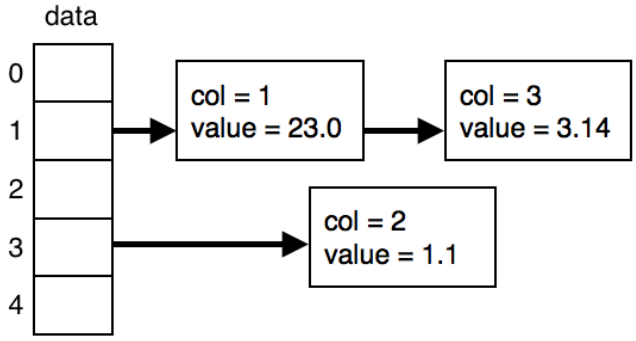
```
class Node {
public:
  Node(unsigned int col, float value) : col_(col), value_(value) {}

  // ACCESSORS
  unsigned int getCol() const {return col_;}
  float getValue() const {return value_;}

  // MODIFIERS
  void putValue(float value) {value_ = value;}

private:
  unsigned int col_;  // the column number
  float value_;
};
```

Here is an illustration of a 5×3 sparse matrix with three non-zero elements. Each row of the data vector contains an STL list of Nodes.



1.1 `Sparse` Class Declaration [/10]

Write the header file for the Sparse class, `sparse.h`.

Your class definition should store the number of rows and columns that the matrix will contain. The constructor should be passed the number of rows and columns that the matrix will contain. You should have accessors for reading the row and column size of the matrix, but not modifiers. The matrix size cannot be changed once it is initialized.

You will need a to include special accessor named `get` which will be passed row and column indices and will return a float value representing the value at those indices in the matrix. Your class should have one modifier named `put`. This method will be passed row and column indices and a value. It will assign the value to the matrix location given by the row and column indices.

Don't worry about `#include` or `#define` statements.

Solution:

```
class Sparse {
public:
  Sparse(unsigned int rows, unsigned int cols);

  unsigned int getNumRows() const {return rows_;}
  unsigned int getNumCols() const {return cols_;}
  float get(unsigned int row, unsigned int col) const;

  void put(unsigned int row, unsigned int col, float value);

private:
  unsigned int rows_;
  unsigned int cols_;
  std::vector<std::list<Node> > data;
};
```

1.2 `Sparse` Class Implementation [/15]

Now implement the constructor, member functions, and any helper functions as they would appear in the `.cpp` file.

The constructor method should have the following behavior:

- the method is passed the number of rows and column that the matrix will contain
- it should initialize a vector with enough elements to hold the lists for each row. Initially the lists will be empty.

The `get` method should have the following behavior:

- the method is passed a row and a column index.
- iterate through the list of columns for the specified row until the specified column is found or it is determined that the node containing the column is not in the list.
- if an element for the specified row and column is found, return the value.
- if an element for the specified row and column is not found, return 0.

The `put` method should have the following behavior:

- the method is passed a row index, a column index, and a value.
- iterate through the list of columns for the specified row until the. specified column is found or it is determined that the node containing the column is not in the list.
- if an element for the specified row and column is found,

  if the value passed to the function is non-zero, replace the value

  if the value passed to the function is exactly zero, remove the node
- if an element for the specified row and column is not found,

  if the value passed to the function is non-zero, insert the node in the appropriate location.

  Maintain the lists in order of increasing column number. Do not sort the lists.

  If the value passed to the function is exactly zero, do nothing.

Solution:

```
#include <iostream>
#include <vector>
#include <list>
#include <cstdlib>
#include "sparse.h"

Sparse::Sparse(unsigned int rows, unsigned int cols) {
  rows_ = rows;
  cols_ = cols;
```

```
  data.resize(rows_);
}

float Sparse::get(unsigned int row, unsigned int col) const {
  if (row >= rows_) {
    std::cerr << "Invalid row " << row <<std::endl;
    exit(1);
  }

  if (col >= cols_) {
    std::cerr << "Invalid column " << col <<std::endl;
    exit(1);
  }

  for (std::list<Node>::const_iterator it = data[row].begin();
       it != data[row].end(); ++it) {
    if (it->getCol() == col)
      return it->getValue();
    else if (it->getCol() > col)
      return 0.0;
  }

  return 0.0;
}

void Sparse::put(unsigned int row, unsigned int col, float value) {
  if (row >= rows_) {
    std::cerr << "Invalid row " << row <<std::endl;
    exit(1);
  }

  if (col >= cols_) {
    std::cerr << "Invalid column " << col <<std::endl;
    exit(1);
  }

  bool inserted = false;
  for (std::list<Node>::iterator it = data[row].begin();
       it != data[row].end(); ++it) {
    if (it->getCol() == col) {
       if (value == 0) {
         data[row].erase(it);
         inserted = true;
      }
      else {
        it->putValue(value);
        inserted = true;
      }
      break;
    }
    else if (it->getCol() > col && value != 0) {
      Node n(col, value);
      data[row].insert(it, n);
      inserted = true;
      break;
    }
  }

  if (! inserted && value != 0) {
    Node n(col, value);
    data[row].push_back(n);
  }
}
```

2 City Distances [/20]

In this problem, you are asked to compute the distances of cities from a point on a map. Data will be read from a file containing city names and their x and y map coordinates. An example of the file is shown to the right. The first line is different from the others: it contains an x coordinate, a y coordinates, and a distance. The remaining lines contain a city name and the x and y coordinates for the city location. In this problem you are asked to find all cities within the given distance from the x and y coordinates given on the first line.

`cities.txt`

```
10 1 5
Troy 10 5
Bravos 30.1 56
Lorath 11 4
Lys 21 8
Pentos 78.2 9
Volantos 1 24.7
```

`std::cout`

```
Troy 4.0
Lorath 3.16228
```

2.1 `City` class [/7]

First create a small class called City to store the city name and x and y coordinates. You should create accessors for private variables, but you do not need to create modifiers. Once created, the city data will not be modified. Write the header file and, if necessary, the implementation file (`.cpp`).

Solution:

```
class City {
public:
  City(const std::string& name, float x, float y) : name_(name), x_(x), y_(y) {}

  const std::string& getName() const {return name_;}
  float getX() const {return x_;}
  float getY() const {return y_;}

private:
  std::string name_;
  float x_;
  float y_;
};
```

2.2 `main` Implementation [/13]

Next, write a `main` function to read the data from the `cities.txt` file. Create City objects, and store them in a vector. Once stored, search the vector and find and print all cities within the given distance from the x and y coordinates read on the first line of file above. Again, don't worry about `#include` or `#define` statements.

Solution:

```
float distance(float x1, float y1, float x2, float y2) {
  return sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
}

int main() {
  std::ifstream istr("cities.txt");
  if (! istr.good()) {
    std::cout << "Can't read cities.txt" << std::endl;
    return 1;
  }

  float point_x;
  float point_y;
  float dist;
  istr >> point_x >> point_y >> dist;

  std::vector<City> cities;
  std::string city;
  float x;
  float y;
  while (istr >> city >> x >> y) {
    cities.push_back(City(city, x, y));
  }
```

```
  for (unsigned int i = 0; i < cities.size(); ++i) {
    float d = distance(cities[i].getX(), cities[i].getY(), point_x, point_y);
    if (d <= dist)
      std::cout << cities[i].getName() << " " << d << std::endl;
  }
}
```

3 Order Notation [/15]

Grading Note: 3 pts each correct answer.

Given the following variables:

```
vector<int> v;
int sum = 0;
```

Assume that `v.size()` is equal to some value, n. Write the order, in O notation, for each of the following:

```
for (unsigned int i = 0; i < v.size(); ++i) {
  sum += v[i];
}
```

Solution: O(n) where n is the size of v

```
for (unsigned int i = 0; i < v.size(); ++i) {
  for (unsigned int j = i; j < v.size(); ++j) {
    sum += v[i] + v[j];
  }
}
```

Solution: O(n^2) where n is the size of v

```
for (unsigned int i = 0; i < v.size(); ++i) {
  for (unsigned int j = 0; j < 2; ++j) {
    sum += v[i] + 7;
  }
}
```

Solution: O(n) where n is the size of v

```
for (unsigned int i = 1; i < n; i *= 2) {
  sum += 7;
}
```

Solution: O(log(n))

```
for (unsigned int i = 0; i < log(v.size()); ++i) {
  sum += 1;
}
```

Solution: O(log(n)) where n is the size of v

4 Debugging [/8]

Grading Note: 2 pts each correct answer.

Each of the following code snippets contains a bug. Indicate what the bug is and describe how to correct it.

```
int* func() {
  int temp = 500;
  return &temp;
}
```

Solution: Returns a pointer to a local variable. Fix: int* temp = new int(500); return temp; or change return type - syntax must be correct

```
int main () {
  char** data = new char*[10];
  for (unsigned int i = 0; i < 10; ++i)
    data[i] = new char[100];
 delete [] data;
}
```

Solution: Memory Leak. Fix: add for (int i = 0; i <10; ++i) delete [] data[i];

```
std::vector<double> v1;
my_vec.push_back(4);
my_vec.push_back(8);
std::vector<int> v2(v1);
```

Solution: v2 is the wrong type. Fix: make types match

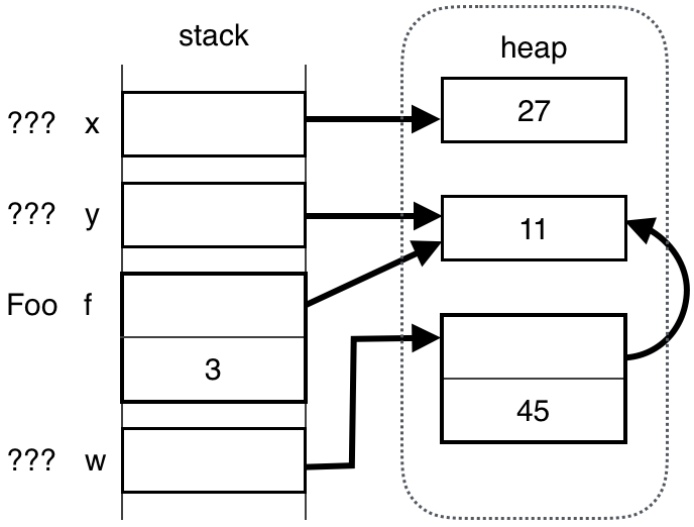
```
int width;
for (unsigned int i = 0; i < 10; ++i) {
  std::cout << std::setw(width) << i*i << std::endl;
}
```

Solution: width is uninitialized Fix: set width to something

5 Memory Diagramming [/20]

Write code to produce the memory structure shown in the diagram to the right. This should include a definition of the class `Foo`. There are 2 Foo objects one on the stack and one on the heap.

Some types have been omitted (marked with "???").



Solution:

```
class Foo {
public:
  Foo(int* w, int t);
private:
  int* w_;
  int t_;
};

Foo::Foo(int* w, int t) {
  w_ = w;
  t_ = t;
}
```

```
int main() {
  int* x = new int;
  *x = 27;
  int* y = new int;
  *y = 11;
  Foo f(y, 3);
  Foo* w = new Foo(y, 45);
}
```

6 iClicker Replay [/9]

Grading Note: -2pts each unanswered or incorrect.

6.1 What is *not* a reason for making the member variables of a C++ class private?

(A) It faciliates future changes to the internal representation to improve efficiency or add new functionality.

(B) This is an *abstraction barrier*, meaning the external user does not need to understand the details of the internal representation.

Solution: C

(C) If you don't specify, everything about a C++ `class` will be public. In contrast, by default, everything is private for a `struct` (inherited from the C programming language).

(D) It is a security measure that prevents accidental or erroneous modification of the object by external users of the class.

(E) Because the instructor said we should always* make them private.

6.2 Which of the following statements is *false* about the keyword `const` and the symbol `&` in C++?

(A) The meaning of the single `&` has nothing to do with the meaning of the double `&&`.

(B) The keyword `const` tells the compiler that we do not intend to modify *something* and then the compiler helps protect against accidental changes to this data.

Solution: E

(C) You should use pass-by-reference when the data you are sending to a function might be big (e.g., STL strings & STL vectors).

(D) When the data is small ($\leq$ 8 bytes), its OJ to pass-by-value (a.k.a. pass-by-copy), unless you want the function to modify it.

(E) Compiler error message related to mismatches in const & reference are long and sometimes confusing to decipher, so it's OK to ignore them.

6.3 Which of the following is *not* a use of the `&` symbol in C++?

(A) To indicate `pass-by-reference` in the argument list of a function.

(B) It means "take the address of" when placed in front of a variable.

(C) `&&` is the *logical and* operator. We use this in `if` statements to check if 2 (or more) things are true.

Solution: D

(D) It means "follow that pointer" when placed in front of an object of pointer type.

(E) It prevents unnecessary copying of large data structures when passing data to or returning data from functions. (But it must be used appropriately.)

6.4 Which of the following statement(s), if any, is not an important consideration when increasing the capacity of an STL `vector` object when we call `push_back`?

(A) Memory allocation sizes that are powers of two are more likely to align with virtual memory pages, cache boundaries, etc.

(B) If we only make the new array 1 element larger than the old array, we will have to do another expensive resize & copy on the very next `push_back` to this vector object.

Solution: E

(C) The most efficient solution is to know/determine how many elements we will need to store and allocate exactly that amount of memory at the start.

(D) If we resize the vector to be significantly larger than necessary, we will waste memory.

(E) Everything is important.

6.5 Which of the following statements is true about heap vs. stack allocation?

(A) Arrays are always allocated on the heap.

(B) Heap memory associated with a variable automatically goes away when the scope for a variable goes away when the scope (curly braces "{ }") containing that variable closes.

Solution: D

(C) If you forget to use the "`delete`" keyword on heap allocated memory, you will permanently lose those locations of memory and never be able to reuse them. Even after rebooting your computer!

(D) If your code (or a helper function or library function) doesn't explicitly use the "`new`" keyword, your data will always be on the stack or in the data section.

(E) We should allocate our big data on the stack, so we can delete it when we are finished with it.