資料結構與程式設計

(Data Structure and Programming)

103 學年上學期複選必修課程 901 31900

Homework #1 (Due: 9:00pm, Oct 7, 2014)

Department:_____ Grade:_____

Id:_____ Name:_____

1. Getting familiar with multi-file project and Makefile.

   Key reviews: *class, data member, member function, C++ string class, header file, Makefile, const, reference variable, pointer variable, array, memory address, sizeof, iostream and its manipulator, debugger, external function.*

   (a) Write a header file "p1a.h" that contains the following class:

```
class P1a
{
public:
   P1a() {}
   P1a(const string& s) : _str(s) {}
   void assign(const string&);
   void print() const;
   P1a& append(const P1a&);

private:
   int    _dummy;
   string _str;
};
```

   Write a C++ file "p1a.cpp" that implements `class P1a`'s member functions:

   (i) The member function "`void assign(const string& s)`" assigns the "`string s`" to the data member "`_str`" by the "=" operator.

   (ii) The member function "`void P1a::print() const`" prints out the data member "`_str`" followed by an "`endl`".

(iii) The member function "`P1a& append(const P1a& p)`" appends the string "`p._str`" to the data member "`_str`" and return a "reference variable of the calling object".

Note that the C++ file "p1a.main.cpp" is included in the homework files and you should not change any part of it.

(b) Implement a C++ file "p1b.cpp" that does the following:

(i) A function[1] "`void printSize()`" to print out the size of `class P1a` as:

```
The size of class P1a is XX.
```

(Replace XX with the actual size)

(ii) A function "`void printStaticArraySize()`" that declares an array of `class P1a`:

```
P1a arr1b_1[5];
```

and prints out the memory addresses of "`arr1b_1[i]'s`" in hex as:

```
============================
Addresses of arr1b_1[5] are:
&arr1b_1[0]: 0x123456ab
&arr1b_1[1]: .....
&arr1b_1[2]: .....
&arr1b_1[3]: .....
&arr1b_1[4]: .....
```

(Replace 0x123456ab and "....." with actual addresses)

(iii) A function "`void printDynamicArraySize()`" that declares an array of `class P1a`:

```
P1a *arr1b_2 = new P1a[5];
```

, prints out the memory addresses of "`arr1b_2[i]'s`" in hex as:

```
============================
Addresses of arr1b_2[5] are:
&arr1b_2[0]: 0x123456ab
&arr1b_2[1]: .....
&arr1b_2[2]: .....
&arr1b_2[3]: .....
```

[1] When not explicitly specified, a "function" means the file-scope function, not a member function.

```
&arr1b_2[4]: .....
```

and prints out the memory address of the variable "`arr1b_2`" in hex as:

```
&arr1b_2: .....
```

(Replace 0x123456ab and "....." with actual addresses)

(iv) A function "`void printPointerArraySize()`" that declares an array of `class P1a` pointers:

```
P1a **arr1b_3 = new P1a *[5];
for (size_t i = 0; i < 5; ++i)
   arr1b_3[i] = new P1a;
```

, prints out the memory addresses of "`arr1b_3[i]'s`" in hex as:

```
============================
Addresses of arr1b_3[5] are:
&arr1b_3[0]: 0x123456ab
&arr1b_3[1]: .....
&arr1b_3[2]: .....
&arr1b_3[3]: .....
&arr1b_3[4]: .....
```

, prints out the contents of "`arr1b_3[i]'s`" in hex as:

```
Contents of arr1b_3[5] are:
arr1b_3[0]: .....
arr1b_3[1]: .....
arr1b_3[2]: .....
arr1b_3[3]: .....
arr1b_3[4]: .....
```

and prints out the memory address of the variable "`arr1b_3`" in hex as:

```
&arr1b_3: .....
```

(Replace 0x123456ab and "....." with actual addresses/values)

Implement a C++ file "p1b.main.cpp" that includes an "`int main()`" that calls all the functions in "p1b.cpp" (with the order as specified above).

(c) Compose a "Makefile" that is able to compile various executables:

(i) Type "make p1a" to compile the source codes in (a). The generated executable should be called "p1a". Execute it and direct the outputs to a file "p1a.out".

(ii) Type "make p1b" to compile the source codes in (b). The generated executable should be called "p1b". Execute it and direct the outputs to a file "p1b.out".

(iii) Type "make bug" to compile the necessary source codes including the C++ file "p1.bug.cpp". The generated executable should be called "p1.bug".

(iv) The default make target should be "p1a" (i.e. by typing "make").

(d) Execute "p1.bug" and you should see the program crashes. Use the debugger "gdb", "lldb" or "ddd" for debugging. Describe how you debug and explain the source/reason of the bug in the file "p1d.out". However, if the program does not crash in your platform, please observe the code and point out if there is any bad coding behavior (that may lead to program crash).

[Warning] A .cpp file may include a .h file, but it should NEVER include another .cpp file. If you need to call a function that is defined in another .cpp file, just declare its function prototype in the beginning of the file ("extern" can be omitted).

2. For the following "`SelectionSort`" function,

```
void selectionSort(vector<int>& array)
{
   for (size_t i = 0, n = array.size(); i < n - 1; ++i) {
      size_t pivot = i;
      for (size_t j = i+1; j < n; ++j) {
         if (!compare(array[pivot], array[j]))
            pivot = j;
      }
      if (pivot != i)
         mySwap(array[pivot], array[i]);
   }
}
```

Implement the following sub-problems under the "p2" directory.

Key reviews: *Standard Template Library (STL) vector, functional object, inherited class, polymorphism, virtual class, pure virtual function, operator overloading, template function,*

(a) Implement the "`main`()" function so that the program can ask the user to input a sequence of numbers and store them in an array. The above function "`SelectionSort`()" will then be called by "`main`()" to sort the numbers in ascending order. Please also implement the functions "`compare`()" and "`mySwap`()" as called in "`SelectionSort`()". However, please DO NOT change the code in "`SelectionSort`()". The following is a sample output:

```
How many numbers? 5
```

```
234
132
532
123
52
Before sort:
234 132 532 123 52
After sort:
52 123 132 234 532
```

For ease of grading, put your code in a single file called "p2a.cpp".

(b) Change the "`compare`()" function into a "*functional object*" argument for the "`selectionSort`()" function. That is, the function prototype will be:

```
void selectionSort
(vector<int>& array, const Compare& compare);
```

Please create another two functional object classes, "`Less`" and "`Greater`", which both inherit the class "`Compare`". Therefore, in "`main`()", if we pass in a functional object of type "`Less`" (i.e. calling "`selectionSort`(*arr*, `Less`())"), the numbers will be sorted in ascending order. On the contrary, the function object of type "`Greater`" will sort the numbers in descending order.

Please make the base functional object class "`Compare`" a virtual class. That is, make its "`operator` ()" a pure virtual function.

Sort the numbers in "`main`()" in both ascending and descending order. A sample output is as follows:

```
How many numbers? 5
234
132
532
123
52
Before sort:
234 132 532 123 52
Ascending sort:
52 123 132 234 532
Descending sort:
532 234 132 123 52
```

For ease of grading, put your code in a single file called "p2b.cpp".

(c) Modify the functions into template functions. That is, the "selectionSort" will not only be able to sort integers, but any data type as long as its "operator <" and "operator >" are defined.

In other words, the function prototype of "selectionSort()" will be:

```
template <class T>
void selectionSort
(vector<T>& array, const Compare<T>& compare);
```

In your "main()", sort a sequence of strings in ascending order, and a sequence of doubles in descending order. A sample output is as follows:

```
How many strings? 5
hello
ric
what
is
up
Before sort:
hello ric what is up
Ascending sort:
hello is ric up what

How many doubles? 6
1.2
3.8
-23.43
-0.02
8.88
10
Before sort:
1.2 3.8 -23.43 -0.02 8.88 10
Descending sort:
10 8.88 3.8 1.2 -0.02 -23.43
```

For ease of grading, put your code in a single file called "p2c.cpp".

3. It is very often that people use pointer variables to avoid the duplication of data. For example, let's say you have a big set of data. On the one hand, you want to store them in an associative data type (e.g. STL's map) so that efficient member queries (e.g. from a string-typed key to look up the corresponding data value) can be assured. On the other hand, you also like to put them in a linear data type (e.g. STL's vector) so that you can quickly iterate all the data and randomly access one of them with a distinct ID. To prevent duplicating data in both containers, we use pointers to share the actual data storage locations. That is, the values stored in both map and vector are the memory addresses of the data (i.e. pointer variables), pointing to the same physical locations for which the data are stored.

   In this problem, our original data source is an arbitrary text file. Once the text file is read in, words are parsed into string tokens and are stored in two distinct data container (in `class AMgr`):

   (1) `AMgr::map<string, A*> _amap`: where the key(string)-value(A*) pair in the map refers to the parsed token (string) and the memory address for the data storage (A*).

   (2) `AMgr::vector<A*> _alist`: where the addresses of the data storage are recorded sequentially into a dynamic array (i.e. vector) in the order as the corresponding parsed tokens appear in the source file.

   The data are actually stored as objects of `class A`, in which there are two data members:

(1) `string _data`: which is the original parsed token.

(2) `vector<Occurence> _occurence`: where `Occurrence` is a *typedef* of `pair<unsigned, unsigned>`. It is to hold the line number and word order of the parsed token as it appear in the text file. Therefore, this data member is to record all the occurrences of the parsed tokens with the same string.

In this homework, main() function and header file are provided as "p3.main.cpp" and "p3.amgr.h". You don't have to modify them. Actually, these two files are included in "`MustRemove.txt`" (See the Homework Rules below). That is, you can't include them in your submission as we will use our version instead.

All the TODOs are in the file p3.amgr.cpp, where you need to:

(1) Implement the constructors (two versions) and destructor of `class A`. Please note that you need to carefully maintain its static data member "`_count`" as the number of *alive objects* of `class A` in the memory.

(2) Initialize the static data member "`_count`" of `class A`.

(3) Implement the function "`AMgr::cleanAll()`" to clean up all the data in `AMgr::_amap` and `AMgr::_alist`.

(4) Realize the operator (<<) overloaded functions for `class AMgr` and `A` for output message. Please see the example below, or refer to "p3.in" and "p3.out.ref" for input sample file and its corresponding output format.

[Example]

Sample source file

```
People are strange when you are a stranger
Faces look ugly when you are alone
Women seem wicked when you are unwanted
Streets are uneven, when you are down
```

Output file

```
Total distinct words: 18
[0] People
( 0, 0 )
[1] are
( 0, 1 )  ( 0, 5 )  ( 1, 5 )  ( 2, 5 )  ( 3, 1 )  ( 3,
5 )
[2] strange
( 0, 2 )
[3] when
( 0, 3 )  ( 1, 3 )  ( 2, 3 )  ( 3, 3 )
[4] you
( 0, 4 )  ( 1, 4 )  ( 2, 4 )  ( 3, 4 )
…
[17] down
( 3, 6 )
```

Note: Extra empty lines are omitted for clarity reason.

Key reviews: *STL map and vector, shared memory by pointer variables, static data member and member function, file stream, string stream, iterator, ostream operator overloading.*

Notes: Please pay attention to the homework rules below and on the website. Failure to abide by the rules may result in deduction in homework points.

[Homework Rules]

1. To save our time in compiling, executing, and automatically grading your homework, please follow the file naming rules as specified in the homework description. There is a Perl script "SelfCheck", included in this homework and also posted on the announcement of Ceiba website, to enforce the rules. Please refer to the README file "SelfCheck_Readme.txt" for the detail usage of "SelfCheck".

2. In short, there are two files, "MustExist.txt" and "MustRemove.txt", for each homework, specifying the files you must turn in and remove before submission. They are used by "SelfCheck" to ensure you have submit the proper files with proper names.

3. To run SelfCheck, please compress your homework with proper file name and then type, for example:

   ```
   ./SelfCheck b77503057_hw1.1.tgz
   ```

   If there is any error, please fix it before submission.

4. In case you can't finish your homework, you may fail "SelfCheck" because there may be some missing files for "MustExist.txt". In that case, you can use the command "touch" to create an empty file in the proper directory in order to pass the SelfCheck. For example, say the file "p2c.cpp" is missing, you can type "touch p2c.cpp" in the directory "p2" to create an empty file.