

# CSIS0396A Programming Methodology and Object Oriented Programming (Class A) Assignment 4

Tutor: gechen@csis.hku.hk, Deadline May 13, 2002, 11:59pm.

Create a CVS repository and do the following two mini-projects in it, using the project name as the CVS directory name. We will read the history recorded in CVS, so make sure that you start using CVS at the beginning of the project. Keep meaningful log message throughout. Handin your CVS repository via the hand-in system of our department.

Be reminded to create an empty directory, check-out all the files from the repository and check whether you can make the programs there before you hand-in.

1. (*Project weakptr: Using templates, 40 marks*) Create a pair of template classes *TObject* and *TWeakPtr*. Conceptually, the classes represent an object of a particular type and a “weak pointer” to it. Both classes should have exactly one template argument, being the type of the object stored. Your *TObject* class should have a constructor which allows a value of type *T* to be used to create a variable of type *TObject<T>*. On the other hand, a *TWeakPtr<T>* is created from a pointer to *TObject<T>* (e.g., *WeakPtr<int>(&obj)*). The weak pointer works like a regular pointer (i.e., the *\**, *->* and *==* operators are defined), except that when the object it points to is deleted, its value becomes NULL automatically; and if a NULL weak pointer is dereferenced, an exception of type *bad\_dereference* is thrown. Your code should look like this:

```
class bad_dereference: public exception ...
template<class T> class TWeakPtr;

template<class T>
class TObject {
    friend TWeakPtr<T>;
public:
    TObject(const T& another) ...
    ~TObject() ...
    T& Get() ... // Get the value of it
private:
    T _impl; // The object itself
    set<TWeakPtr<T>*> _references; // On destruct, set them NULL
};

template<class T>
class TWeakPtr {
public:
    TWeakPtr(TObject<T>* ptr = 0) ...
    ~TWeakPtr() ...
    TWeakPtr& operator=(const TWeakPtr& another) ...
    T& operator*() ...
    T* operator->() ...
    bool operator==(const TWeakPtr& other) ...
private:
    TObject<T>* _object; // What object it points to
};
```

Example code using it:

```
int main() {
    try {
        TObject<int>* po1 = new TObject<int>(10);
        TObject<int>* po2 = new TObject<int>(20);
        TWeakPtr<int> ptr1(po1), ptr2(po2);
        cout << *ptr1 << ', ' << *ptr2 << endl; // 10, 20
        delete po1;
        if (ptr1 == 0) // yes
            cout << "Null pointer" << endl;
        ptr1 = ptr2;
        cout << *ptr1 << ', ' << *ptr2 << endl; // 20, 20
        delete po2;
        cout << *ptr1 << ', ' << *ptr2 << endl; // throw exception
    } catch (bad_dereference&) {
        cout << "WeakPtr null" << endl; // printed
    }
}
```

2. (Project listing: Python programming, 60 marks) Write a Python GUI script that **shows the directory structure** of the filesystem using the Tkinter library. The program should display multiple windows. Within each of them there are two listings, one showing the directory structure, and the other showing the files within a chosen directory. There is also a one-line status bar at the bottom of the window. At any time, the information about **one** directory *d* is shown. When a new window is created, *d* is the current directory, but this can be changed as described below. The details:

- The directory list shows the names of all directories from the root to the directory *d*, and all sub-directories immediately within these directories. The names of all (immediate) sub-directories of the same directory should appear in sorted order, while the names of the sub-directories of a directory should appear immediately after the directory. The names are indented according to the depth of the directory: the root directory appear with no indentation, and each deeper level has 3 more spaces as indentation. If you click on one of these labels, the directory *d* should be set to this directory. Filesystem information of *d*, including the number of mega-bytes in the filesystem and the number of mega-bytes available for use, should then appear in the status bar.

*Implementation notes:* When a directory is clicked on, some directory disappears, and some appears. The easiest way to implement it is to get all the names of the directories again. However, care should be taken to set the list so that the name of the current directory remain unchanged on the screen, if possible.

*Exceptional case:* What if the directory the user clicks on is no longer there? Then the program should display an error message on the status bar, with *d* set to the first ancestor of the chosen directory which does exist.

- The file list should contains the names of all **non-directory entries** of the directory *d*, again in alphabetical order. If the user click on one of these names, the command “file” should be invoked on the file, and the result should be displayed on the status bar..
- Finally, there should be a menu bar, allowing the user to **open one more window** containing similar information (but may be viewing another directory), and to **Quit the program** (i.e., close all windows).

**Hints:**

- Read the documentation of the Python modules “string”, “os”, “os.path” and “commands”.
- The names of the entries in the list box is not sufficient to determine what directory to use. Instead of guessing the whole directory path, store the full path of each directory in an array for each window.
- You need to manually bind an event to the “Listbox”. Instead of using an event like “<Button-1>”, use the virtual event “<<ListBoxSelect>>”. It happens after the selection is set, so you can get the current selection in the event handler.
- It is quite a challenge to keep the directory entry in the same place before and after clicking on a directory name. To do so, remember the line number of the entry before changing the directory entries, and find the line number of the same entry after changing the directory entries. You can then derive information about whether you want to shift the entry up, by calling yview method of Listbox.
- You’ll have to handle quite some data in each window: a set of widgets, the directory full paths, etc. You probably want to have an object storing all these data, since there are multiple instances of them. The easiest way is to derive a class from Toplevel, and add all these information to the object as data member.
- To avoid the possibility to close the main window (and thus quit the program accidentally), the main window should be withdrawn from the screen at the beginning using the “withdraw” method.
- Once the (withdrawn) main window becomes the only window, you have to quit the program. To do this, you probably needs to keep a count on the number of Frames that are created but not closed. You can do this by marking a Frame as deleted when a <Destroy> event is sent to it.
- Be very careful about invoking the “file” command: it is possible that the user clicks on a file that contain a special character like a single quote or a dollar sign. If you use *commands.getoutput*, the string you give it is passed to the shell program, which interpret these characters. You have to escape them by adding a back-slash in front of each of these characters.

**Instruction to hand-in the repository:** change directory to the repository (not working directory!), and type “tar czvf ~/a4.tgz listing weakptr”. Then hand-in the file a2.tgz in your home directory.