CSE687 Midterm #3

Name: ___________________________________ SUID: _____________

This is a closed book examination. Please place all your books on the floor beside you. You may keep one page of notes on your desktop in addition to this exam package. All Exams will be collected promptly at the end of the class period. Please be prepared to quickly hand in your examination at that time.

If you have any questions, please do not leave your seat. Raise your hand and I will come to your desk to discuss your question. I will answer all questions about the meaning of the wording of any question. I may choose not to answer other questions.

You will find it helpful to review all questions before beginning. All questions are given equal weight for grading, but not all questions have the same difficulty. Therefore, it is very much to your advantage to answer first those questions you believe to be the easiest.

1. Describe what happens when a C++ package is compiled.

   The preprocessor reads the package implementation (.cpp) file, building a composite temporary source file for the compiler. Each of the #includes causes the text of the included file to be placed inline at the site of the include. Each #ifdef - #endif causes code to be included or not included in the temporary source file. Each #ifnotdef #define - #endif causes an included header to be included only once.

   The compiler processes one implementation file at a time and, if there are no translation errors, generates an .obj file for each one. If there are errors, error messages are emitted and compilation of that .cpp ends.

   Subsequently the linker will bind all the obj files and any cited libraries into an executable file, static library, or dynamic link library. Should any executable build include a DLL lib file, the loader will link that to the executable image at load time, or optionally at run-time due to an explicit load library command.

2. Write a class declaration for the Abstract Syntax Tree used in Project #2. Please also write a member function that traverses the tree applying a callable object supplied as a function argument to each node of the three. Please state any assumptions you need to make about the tree nodes in order to write this function.

   Assumption:
   AST Nodes are stored in the native heap and have destructors that call delete on their children.

```
template <typename Node>
class AST
{
public:
  AST() {}
  AST(const AST&) = delete;
  AST& operator=(const AST&) = delete;
  Node*& root();
  template <typename CallObj>
  void walk(CallObj& co);
  template <typename CallObj>
  void walk(Node* pNode, CallObj& co);
  size_t indent() { return _indent; }
  virtual ~AST() { delete _pRoot; }
private:
  ASTNode* _pRoot = nullptr;
  size_t _indent = 0;
};

template<typename Node>
Node*& AST<Node>::root()
{
  return _pRoot;
}

template <typename Node>
template <typename CallObj>
void AST<Node>::walk(Node* pNode, CallObj& co)
{
  co(pNode);
  _indent += 2;
  for (auto ptrNode : pNode->children)
    walk(ptrNode, co);
  _indent -= 2;
}

template<typename Node>
template<typename CallObj>
void AST<Node>::walk(CallObj& co)
{
  _indent = 0;
  walk(_pRoot, co);
}
```

3. What are the advantages of using a lambda in place of a function?

   Lambdas are a quick way to write the equivalent of a functior:

   a. Lambdas can be defined locally, including inside a function, so that it is obvious what happens when invoked.
   b. They can be stored or enqueued for later execution – see the Enqueued WorkItems in CppThreadTechniques.
   c. They can be passed to and returned from functions.
   d. They can carry their arguments as captured variables.

4. State the Open/Closed Principle and describe how to apply it in practice. Give an example discussed in class where OCP has been applied.

   "Software entities (classes, packages, functions) should be open for extension but closed for modification."

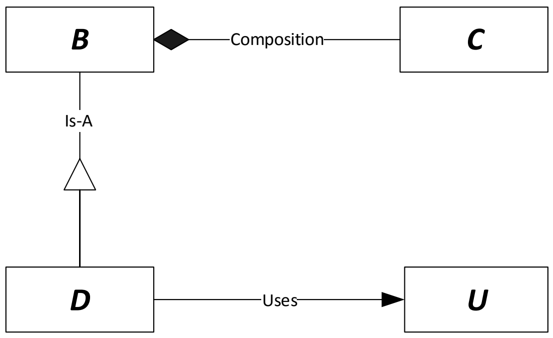
   It doesn't make much sense to apply OCP to Application-Side software. But for Solution-Side software we expend a lot of effort to make code reusable, and that is where we expect to apply OCP. We do that by building template classes, Inheritance hierarchies, and using Hooks and Mix-ins.

   OCP applies to the parser – we extend parsing activities by adding new derived rules and new derived actions. See the parser example in MTCode Supplement.

   It also applies to the FileMgr as implemented in the MTCode Supplement. The FileMgr there provides two hooks for application specific event handling when files and when directories are encountered. Thus FileMgr is reusable (closed) and is extended by the new event handlers.

5. Given the compound object shown in the diagram, below, what happens when the following code executes?

```
D d1;
D d2 = d1;
```



The composite object d constructed from D contains, within its memory footprint, an instance of the base class B which in turn contains an instance of the composed class C.

Construction of D will always cause B to be constructed (whether we explicitly invoke the B constructor or not) and construction of B will always cause C to be constructed (whether we explicitly invoke the C constructor or not).

Designers can always control how construction proceeds by explicitly invoking constructions on bases and member instances in constructor initialization sequences.

For default construction there is no need to do that as the compiler will always invoke default constructors on bases and members.

For copy and move constructions the designer must explicitly invoke the base and member constructors.

See code in MT3Q5.cpp for these details.

/////// full answer starts here //////////////////////////

First statement is a default construction of D which causes default construction of B and C.
For copy construction in D d2 = d1, the composite object and its parts B and C are all copy constructed, since the code should define initialization sequences that do that or allow compiler to generate copy constructors for C, B, and D if they are correct.

Since U is used but not owned by D, none of the D operations directly causes U to be created or destroyed.

6. Suppose we have a FileMgr class that, given a path and a set of patterns, navigates the directory subtree rooted at the path, and finds all the files matching any of the patterns. How do you design it so that applications can use FileMgr without putting any application specific code in the FileMgr package?

Method #1 – use events and put application code in event handlers

FileMgr provides two hooks, IFileEventHandler and IDirEventHandler. Applications derive from these and register the derived instances using FileMgr::regForFiles(IFileEventHandler*) and FileMgr::regForDirs(IDirEventHandler*).

When a file is discovered by FileMgr it invokes its collection of IFileEventHandlers like this:
pEvtHandler->execute(fileSpec);

When a directory is discovered by FileMgr it invokes its collection of IDirEventHandlers like this:
pEvtHandler->execute(dirSpec);

See FileMgr project in MTCodeSupplement-S16 for all the details.

Method #2 – put application code in derived class virtual function overrides

An alternate method is to make virtual functions, say virtual file(const std::string& fileSpec) and virtual dir(const std::string& dirSpec) in FileMgr that pass the fileSpec or directorySpec when a new file matching one of the patterns is discovered or a new directory is entered. FileMgr should also declare its destructor as virtual.

An application can simply derive from FileMgr and override the the virtual methods to do application specific stuff. We don't need to make the base FileMgr application specific at all.

This way you get to reuse the directory navigation code with no change.

Note that you cannot simply have a FileMgr function return a collection of all the results. That is certainly independent for applications where that will work. However, for applications like Project #3, where you have to enqueue fileSpecs as you find them, it will not work.

7. Write a function that will create a child thread using the same function. Is there a problem with that? If so, can you write code to avoid the problem? Why would you write such a function?

   If you don't provide some way to limit the number of threads, the program will keep creating threads until it runs out of stack memory.

```
void threadProc(size_t count)
{
  std::cout << "\n  entered threadProc with count = " << count;

  // do setup for this thread, probably
  // different from child thread's setup
  std::this_thread::sleep_for(std::chrono::milliseconds(100));

  if (count++ < 10)
  {
    std::thread t(threadProc, count);
    t.detach();
  }
  // do this thread's work here, perhaps in a
  // long-running loop

  std::cout << "\n  leaving threadProc with count = " << --count;
}

using namespace Utilities;
using Utils = StringHelper;

void main()
{
  Utils::Title("MT3Q7 - Recursive thread creation");
  std::thread t(threadProc, 1);
  t.detach();

  std::cout << "\n  press any key to exit: \n";
  _getche();
  std::cout << "\n\n";
}
```

   You write a threadProc like this when you want to create threads in a specific sequence, probably doing some setup needed for each. Note that you are guaranteed that each thread starts running before the next thread is created.

   See MTCode-S16, MT3Q7.cpp for more details.