CSE687 Midterm #2

Name: ____Instructor's Solution_____ 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. Write a class declaration for a scope stack that holds instances of an unspecified type. It should support traditional stack behavior, but also support indexing of, and iterating over, its contents.

   Answer:

```
template<typename T>
class ScopeStack
{
public:
  using iterator = typename std::deque<T>::iterator;
  using const_iterator = typename std::deque<T>::const_iterator;

  void push(T);
  T pop();
  T top();
  Void clear();
  size_t size();
  T& operator[](size_t i);
  T operator[](size_t i) const;
  iterator begin();
  iterator end();
  const_iterator begin() const;
  const_iterator end() const;
private:
  std::deque<T> stack_;
};
```

   Could use **std::vector<T>**, but **std::deque<T>** makes it simpler to iterate front to back.
   Can't use std::stack<T>. That does not support indexing.
   Don't need copy or move ctor or copy or move assignment, or destructor. Compiler generated operations are correct.

   Complete implementation in MTCode/MT2-Q1.

2. Write code for a thread that searches a text file for a specified string[1] and safely returns a Boolean value that indicates whether the string is contained at least once in the file. You may wish to start by assuming you have a synchronous function:

```
bool textSearch(const std::string& fileSpec, const std::string& text)
```

Write the threading code, then, if you have time later, fill in the code for this function.

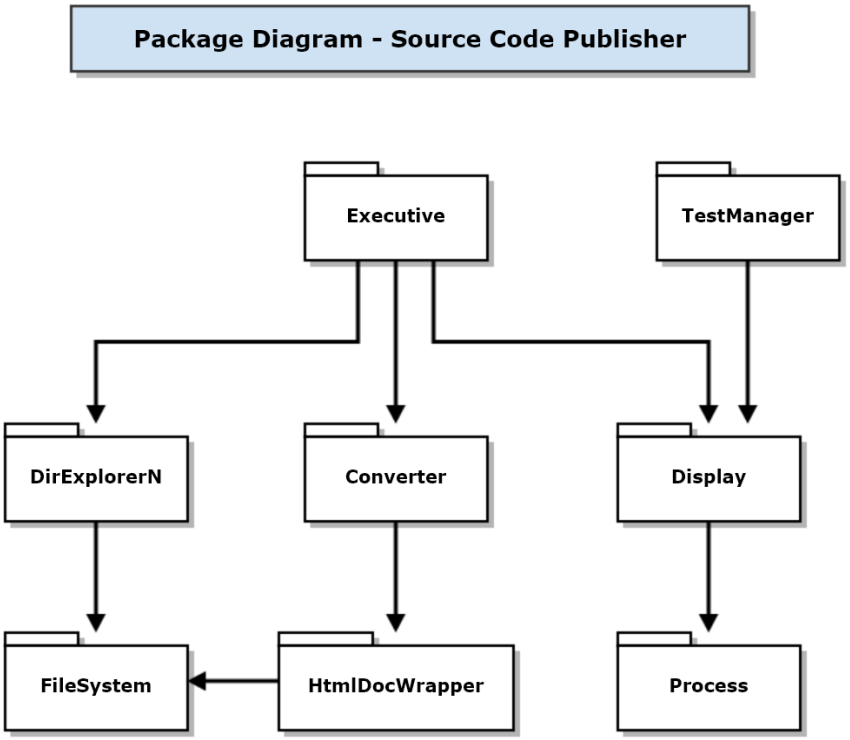
Answer:

```
bool textSearch(const std::string& fileName, const std::string& text)
{
  std::ifstream in(fileName);
  if (!in.good())
  {
    std::cout << "\n  can't open file \"" << fileName.c_str() << "\"";
    return false;
  }
  std::string line;
  while (std::getline(in, line).good())
  {
    if (line.find(text) != std::string::npos)
    {
      in.close();
      return true;
    }
  }
  in.close();
  return false;
}
int main()
{
  std::cout << "\n  MT2-Q2 - thread searches text file";
  std::cout << "\n ------------------------------------\n";

  std::string fileSpec = "../MT2-Q1/MT2-Q1.cpp";
  std::string text = "CSE687";

  bool result;
  std::function<void(const std::string&, const std::string&)> sf =
    [&result](const std::string& fileSpec, const std::string& str) {
    result = textSearch(fileSpec, str);
  };
  std::thread thrd(sf, fileSpec, text);
  std::cout << "\n  main thread started search thread";
  thrd.join();

  if (result == true)
    std::cout << "\n  found \"" << text << "\" in " << fileSpec;
  else
    std::cout << "\n  did not find \"" << text << "\" in " << fileSpec;
```

```
Pseudo Code:
textSearch function
  Open file and read each line
  Search line for text and return true if found
  Return false
main function
  Create lambda that runs function and sets
  captured Boolean to value of search.
  Start thread with lambda and parameters,
  Join thread, then report results.
```

[1] Note that this capability is more useful if the thread can accept and use a regular expression.

3. Draw a package diagram[2] for your implementation of Project #1.



[2] You don't need to show packages for the parser or any of its parts.

4. What does the code in the box do? You may assume the code compiles, and are required to describe everything that happens when the code executes.

```
template <typename T>
class P {
public:
    P(T t) : t_(t) {}
    T operator()() { return t_; }
    void operator()(T t) { t_ = t; }
private:
    T t_;
};
```

```
P<double> p(3.14159);
std::cout << p();
p(0.3333);
std::cout<< p();
```

Answer: Note – method arguments passed by value

a. t_ is initialized with double using P(T t) and T copy ctor
b. Copy of t_ is sent to std::cout using operator()() and T copy ctor
c. Copy of new value is assigned to t_ using operator()(T t) and T move or copy assignment
d. Copy of t_ returned using operator()() and sent to std::cout

P<T> has a compiler generated copy constructor, but it is never called.

```
using P<double>
----------------
ctor P(T t) invoked
operator()() invoked
3.14159
operator()(T t) invoked
operator()() invoked
0.3333
```

```
using P<Wrapped<double>> to show details
-----------------------------------------
Wrapped promoted 3.14159
3.14159 copied
ctor P(T t) invoked
Wrapped destroyed

operator()() invoked
3.14159 copied
Wrapped cast
3.14159
Wrapped destroyed

Wrapped promoted 0.3333
operator()(T t) invoked
Wrapped copy assigned
Wrapped destroyed

operator()() invoked
0.3333 copied
Wrapped cast
0.3333
Wrapped destroyed

dtor ~P() invoked
Wrapped destroyed

dtor ~P() invoked
```

5. Write all the code for a class that behaves like a std::string, but has an additional method:

```
title(const std::string& theTitle)
```

That method writes to the console the title string and, on a succeeding line, a series of hyphens "-" that have the same length as the title string. Can you think of any issues with your design?

Answer:

```
class stringEx : public std::string
{
public:
  stringEx(const char* sPtr) : std::string(sPtr) {}
  stringEx(const std::string& str) : std::string(str) {}

  // compiler defined copy ctor, copy assignment, and destruction
  // are correct so we don't supply them

  void title()
  {
    std::cout << "\n  " << *this;
    std::cout << "\n " << std::string(this->size() + 2, '-');
  }
};
```

std::string does not have a virtual destructor, so we should not call delete for a stringEx instance using a std::string pointer.

6. State the Dependency Inversion Principle. Why is it important, and where have you used it in your design of Project #3 (changed to #2 at beginning of exam)?

   Answer:

   High-level components should not depend on low-level components. Instead, both should depend on abstractions.

   DIP shows us how to solve the problem of weak semantic separation of class interface from its implementation. When a class implementation changes its declaration must change in a consistent way. By hiding the implementation behind an interface and object factory we have isolated the implementation from the client.

   To implement DIP you must have an abstract interface[3] and an object factory that creates an instance of a class that implements the interface and returns a pointer to it, typed as a pointer to the interface.

   DIP was used in Semi with ITokenCollection and the configureParser factory. Similarly, parser uses IRule and IAction interfaces and the configureParser factory.

   The class diagram in my solution for MT1-Q4 shows other places that DIP can be used effectively in Projects #1 and #2.

[3] In C++ a class is abstract if it declares pure virtual functions or if it derives from a base with pure virtual functions and does not provide a definition for the pure virtual functions.

7. What are the benefits of using templates? Where, in your Project #1 code, have you used templates, either of your own design, or from some other source.

   Answer:

   Templates support large-scale code reuse for all classes that act on more than one type in the same way. STL containers are a great example.

   Templates also allow us to build functions and class methods that have parameters that bind, at compile-time, to many different types of objects. That provides great flexibility for many types of implementations.

   Template policies allow us to configure classes for application specific needs, e.g., to lock or not to lock, how to create instances, how to handle events, e.g., DirExplorerT uses a template policy to provide application specific processing for the new file and new directory events.

   Templates were used in Project #1 in several utilities, and, if you used it, in DirExplorerT. You used templates wherever you used STL containers. Another good place to use templates is in an HTML document generator that inserts the webified code into an HTML document. We could define several document structures with different styles by supplying template arguments, like this:

```
HtmlDocGenerator<MyCodeStyle> docGen;
Parser.h.html = docGen.insert(Parser.h.convert);
```

   If the temporary internal representation is a string, holding both the document markup and the converted source text, then the return would use an efficient string move.