Homework 02 Solutions

ECE 449/590, Fall 2022

1. *(10 points)* Consider the following function `access_element_by_index` that returns the iterator to the element at the index `i` in the list `l`.

```
typedef std::list<int> int_list;
int_list::iterator access_element_by_index(size_t i, int_list &l)
{
    assert( ? );
    ...
}
```

   Assume the first element of the list is at the index 0, the second at the index 1, and so on. If the function is required to return an iterator that can be dereferenced, determine the precondition of the function and write an assertion to validate it. (You don't need to implement the function.)

   Answer: `assert(i < l.size());`

2. *(20 points)*

   Consider a container of type `std::map<std::string, int>`. A C++ function `find_or_throw` will search a value from one such container given a key. It will return the value if the key exists and throw `std::runtime_error` otherwise. Design the function interface (parameters and return type) and implement the function body.

   Answer:

```
int find_or_throw(
    const std::map<std::string, int> &c,
    const std::string &key)
{
    auto it = c.find(key);
    if (it != c.end())
        return it->second;
    else
```

```
        throw std::runtime_error("No such key!");
}
```

3. *(20 points)* For the course project, two classes `program` and `evaluation` are used at the top level to interact with the Python code. If we would like to support multiple C++ implementations for `evaluation`, e.g. a simple implementation, a second implementation using multi-threading, and a third implementation using CUDA, how would you modify the current class design so that the changes would be minimal? Describe your ideas without providing implementation details.

   Answer: One idea is to make `evaluation` an interface. The `program` will make use of a factory to create a desired implementation of `evaluation`. If you would like your Python code to choose what implementation to use, an additional parameter of string type can be passed to C++ code for the factory to choose the implementation.

4. *(30 points)* Assume there is no compiling or linking error. Review the following pieces of code and briefly explain potential issues.

   A.
   ```
   std::string &get_hello() {
       std::string s = "hello";
       return s;
   }
   ```
   Answer: UB as a reference to the local variable `s` is returned.

   B.
   ```
   class time {
       int hour, min, sec;
   public:
       bool set(int h, int m, int s);
       int get_hour() const;
       int get_min() const;
       int get_sec() const;
   };
   ```
   Answer: There is no ctor. So the compiler will generate a default ctor, which will NOT initialize the members of built-in types, i.e. `hour`, `min`, `sec`. This may lead to UB at a later time.

   C.
   ```
   class collection {
   };
   class my_array : public collection {
       std::vector<int> vec_;
   };
   void test() {
       collection *p = new my_array;
       delete p;
   ```

```
}
```

Answer: `collection` doesn't have a virtual dtor. Therefore, when `delete p`, the dtor of `collection` instead of dtor of `my_array` will be used, leading to memory leakage.

5. *(20 points)* Consider the classes `base` and `derived` as follows.

```
class base {
protected:
    virtual void step_one() {std::cout << "base::step_one" << std::endl;}
    virtual void step_two() {std::cout << "base::step_two" << std::endl;}
public:
    void run() {
        std::cout << "enter base::run" << std::endl;
        step_one();
        step_two();
        std::cout << "exit base::run" << std::endl;
    }
};

class derived : public base {
protected:
    void step_one() {std::cout << "derived::step_one" << std::endl;}
    void step_two() {std::cout << "derived::step_two" << std::endl;}
public:
    void run() {
        std::cout << "enter derived::run" << std::endl;
        step_one();
        step_two();
        std::cout << "exit derived::run" << std::endl;
    }
};
```

   A. What's the output of the following function `test1`?

```
void test1() {
    derived d;
    derived *p = &d;
    p->run();
}
```

Answer:

```
enter derived::run
derived::step_one
```

```
derived::step_two
exit derived::run
```

B. What's the output of the following function `test2`?

```
void test2() {
    derived d;
    base *p = &d;
    p->run();
}
```

Answer:

```
enter base::run
derived::step_one
derived::step_two
exit base::run
```