Polytechnic Tutoring Center

Final Exam Review ANSWER KEY
CS 2124, Fall 2021

Disclaimer: This mock exam is only for practice. It was made by tutors in the Polytechnic Tutoring Center and is not representative of the actual exam given by the CS Department.

1.

```
class A {
private:
	string aString;
protected:
	const string& get_string() const {
		return aString;
	}
public:
	A(const string& anotherString) : aString(anotherString) {}
};

class B : public A {
public:
	B(const string& otherString) : A(otherString) {}
	void display_string() const {
		cout << get_string() << endl;
	}
};

class C : public A {
public:
	C(const string& anotherotherString) : A(anotherotherString) {}
	void display_string() const {
		cout << aString << endl;
	}
};

int main() {
	A a("A");
	B b("B");
	C c("C");
	b.display_string();
	c.display_string();
}
```

What is the result of the above program?

a. Runtime error as a result of Class B's display_string method
b. Runtime error as a result of Class C's display_string method
c. The output: B
   C
d. **Compilation error as a result of Class C's display_string method**
e. Compilation error as a result of Class B's display_string method

2.

What does the following code result in? And what key concept does this type of code allow?

```
class A {
private:
	string aString;
public:
	A(const string& anotherString) : aString(anotherString) {}
};

class B : public A {
public:
	B(const string& otherString) : A(otherString) {}
};


int main() {
	A* b = new B("B"); \\ line A
	B* a = new A("A"); \\ line B
}
```

a. Results in a compilation error at line A; polymorphism
b. **Results in a compilation error at line B; polymorphism**
c. Results in a compilation error at line A; principle of substitutability
d. Results in a compilation error at line B; principle of substitutability

3. What will be the output of the following code?

```
class A {
private:
	string aString;
protected:
	const string& get_string() const {
		return aString;
	}
public:
	A(const string& anotherString) : aString(anotherString) {}
	~A() { cout << "~A()\n"; }
};

class B : public A {
public:
	B(const string& otherString) : A(otherString) {}
	void display_string() const {
		cout << get_string() << endl;
	}
	~B() { cout << "~B()\n"; }
};
```

```
int main() {
	A* a = new B("B");
	B* b = new B("B2");
	delete a;
	delete b;
}
```

a. ~A()
   ~B()

b. **~A()**
   ~B()
   ~A()

c. ~B()
   ~A()
   ~B()

d. ~B()
   ~A()
   ~B()
   ~A()

e. ~A()
   ~B()
   ~A()
   ~B()

4. Which of the following is not one of the “containers” defined in the Standard Template Library?

   a. Deque
   b. Stack
   c. Queue
   d. Vector
   e. **Graph**
   f. Multiset
   g. Set
   h. Multimap

5. What is the result of the following code? Output on the line below…

```
vector<int> vi = { 10, 2, 4, 1, 5, 3, 7, 9, 8, 6 };
list<int> li(vi.begin(), vi.end());
for (int elem : li) {
	cout << elem << " ";
}
```

10 2 4 1 5 3 7 9 8 6

---

6. Write a templated function that enables you to find a specified value within any iterable object containing only objects of one type. Note this "specified value", along with the beginning and end should be passed into the function. If you couldn't find the value, return a reasonable value.

```
template <typename V, typename U>
V templatedFind(V begin, V end, U needle) {
	for (V it = begin; it != end; ++it) {
		if (*it == needle) {
			return it;
		}
	}
	return end;
}
```

7. Write a function that recursively identifies whether the string passed into the function is a palindrome. Note: you are allowed to use helper variables like the typical high and low frequently seen in data structures.

```
bool is_palindrome(const string& aString, size_t front, size_t back) {
	if (front >= back) {
		return true;
	}
	else {
		if (aString[front] == aString[back]) {
			return is_palindrome(aString, front + 1, back - 1);
		}
		else {
			return false;
		}
	}
}
```

8. Write a function which recursively identifies the number of 1s in the binary representation of a provided integer.

```
int num_ones(int n) {
	if (n == 0) {
		return 0;
	}
	else {
		int x = num_ones(n / 2);
		if (n % 2 == 1) {
			return x + 1;
		}
		else {
			return x;
		}
	}
}
```

9. Write a Iterator class that iterates over a singly linked list. Don't worry about const vs non-const iterators. Just assume the iterator class you are writing is of the non-const type. Write the pre and post increment operators as well as the operator== and operator!= operators for the iterator. Assume the class has the following Node struct definition…

```
struct Node {
	Node* next;
	int data;
	Node(Node* next = nullptr, int data = 0) : next(next), data(data) {}
};
```

```
class LinkedListIterator {
private:
	Node* p;
public:
	LinkedListIterator(Node* ptr) : p(ptr) {}
	LinkedListIterator& operator++() {
		p = p->next;
		return *this;
	}
	LinkedListIterator operator++(int) {
		LinkedListIterator it = *this;
```

```
            ++(*this);
            return it;
        }
        Node operator*() {
            return *p;
        }
        bool operator==(const LinkedListIterator& another_it) const {
            return p == another_it.p;
        }

        bool operator!=(const LinkedListIterator& another_it) const {
            return !(*this == another_it);
        }
};
```