ECE326 – Fall 2019: Week 9 Exercise Questions

1. True or False [1 mark each]

Circle T is true, otherwise circle F for false.

1. Type traits in C++ is an example of reflective programming. **(T)** **F**

   Static introspection is part of reflective programming.

2. Unevaluated context means the expression cannot be evaluated (i.e. is ill-formed). **T** **(F)**

   It means the expression is only used at compile time and not evaluated at runtime.

3. To use an integer vector iterator inside a template, the type of the iterator is specified as `typename vector<int>::iterator`. **(T)** **F**

4. C++ type checks code even if it is unreachable. **(T)** **F**

5. Non-template functions that take variable number of arguments have precedence over base template functions that accepts any type T. **T** **(F)**

2. rm_const **[5 marks]**

Complete the implementation of rm_const, which remove const from type T if T is const-qualified (meaning that const is part of T, e.g. const int).

```
// becomes int
cout << std::is_const<rm_const<const int>::type>::value << endl;

// still int
cout << std::is_const<rm_const<int>::type>::value << endl;
```

See isptr.cpp

3. is_ptr **[7 marks]**

Complete the implementation of is_ptr, which checks whether the type T is a pointer or not. You must be able to handle a const pointer (e.g. int * const). Hint: rm_const may be helpful.

```
template<typename T> is_ptr;

cout << is_ptr<int>::value << endl;          // 0
cout << is_ptr<int *>::value << endl;        // 1
cout << is_ptr<int **>::value << endl;       // 1
cout << is_ptr<int &>::value << endl;        // 0
cout << is_ptr<int * const>::value << endl;  // 1
```

See isptr.cpp

4. can_equate **[5 marks]**

Write a C++ template, can_equate, which checks whether type T can be equal to type U. You must handle the situation when either T or U overloads the == operator.

See equate.cpp