quiz2. Code=1 Digipen login:_____

1. **Problem** (9 * 1 pts):

   Given the following definitions

```
+-----------------+-------------------------+-------------------+
| class B {       |    class D : public B { |    class C {      |
|   public:       |      public:            |      public:      |
|     void f1();  |        void f4();       |        void f5(); |
|   protected:    |    };                   |      private:     |
|     void f2();  |                         |        B b;       |
|   private:      |                         |    };             |
|     void f3();  |                         |                   |
| };              |                         |                   |
+-----------------+-------------------------+-------------------+

D d;
C c;
```

   determine whether each of the following statements/functions compiles or not.

A) compiles B) does not compile	1-1._____ `d.f1();` 1-2._____ `d.f2();` 1-3._____ `d.f3();` 1-4._____ `c.f1();` 1-5._____ `void D::f4() { return this->f2(); }` 1-6._____ `void D::f4() { return this->f3(); }` 1-7._____ `void C::f5() { return b.f1(); }` 1-8._____ `void C::f5() { return b.f2(); }` 1-9._____ `void C::f5() { return b.f3(); }`

2. **Problem** (7 * 3 pts):

   Which of 7 methods below compile? Notice that they are all syntactically sound.

```
class C {
  public:
    C() { data = new int (100); }
    ~C() { delete data; }
    int        GetInt()         const { return *data; }
    int&       GetRefInt()      const { return *data; }
    const int& GetRefConstInt() const { return *data; }

    int*         GetPtr()         const { return data; }
    int*&        GetRefPtr()      const { return data; }
    const int*&  GetRefPtrConst() const { return data; }
    int* const & GetRefConstPtr() const { return data; }
  private:
    int * data;
};
```

A) compiles B) fails	2-1._____ GetInt() 2-2._____ GetRefInt() 2-3._____ GetRefConstInt() 2-4._____ GetPtr() 2-5._____ GetRefPtr() 2-6._____ GetRefPtrConst() 2-7._____ GetRefConstPtr()

3. **Problem** (6 * 1 pts):

Given the following definitions

```
+--------------------------+-------------------------------+
| class B {                |     class D : public B {      |
|   public:                |       public:                 |
|     virtual void f1();   |         void f1();            |
|              void f2();  |         void f2();            |
| };                       |         void f3();            |
|                          |     };                        |
+--------------------------+-------------------------------+
```

```
B b;
D d;
B *pb1 = &b, *pb2 = &d;
```

corresponding function of which class is called for each of the following statement, choose NC if does not compile.

A) `D::fx()` B) `B::fx()` C) `NC`	3-1._____ `b.f2();` 3-2._____ `pb1->f3();` 3-3._____ `pb1->f2();` 3-4._____ `pb1->f1();` 3-5._____ `pb2->f3();` 3-6._____ `d.f3();`

4. **Problem** (5 * 2 pts):

Given the following definitions

```
class B            { public: int i; };
class D : public B { public: int j; };
int foo0(B  b) { return b->j;  }
int foo1(B  b) { return static_cast<D*>(&b)->j;  }
int foo2(B& b) { return static_cast<D*>(&b)->j;  }
int foo3(B* b) { return static_cast<D*>(b)->j; }
int foo4(B* b) { return dynamic_cast<D*>(b)->j; }
D d;
d.i=1;
d.j=2;
```

What happens in each of the following cases:

A) OK B) compiles, but possible run-time error C) does not compile	4-1._____ `foo0(d);` 4-2._____ `foo1(d);` 4-3._____ `foo2(d);` 4-4._____ `foo3(&d);` 4-5._____ `foo4(&d);`

5. **Problem** (7 * 2 pts):

Given the definitions and paragraph from C++ standard:

```
/*
 * 14.8.2.1 Deducing template arguments from a function call [temp.deduct.call]
 *
 * Template argument deduction is done by comparing each function template parameter
 * type (call it P) with the type of the corresponding argument of the call (call it A)
 * as described below.
 *
 * If P is not a reference type:
 * -- If A is an array type, the pointer type produced by the array-to-pointer
 *     standard conversion (4.2) is used in place of A for type deduction; otherwise,
 * -- If A is a cv-qualified type, the top level cv-qualifiers of A's type are
```

```
 *    ignored for type deduction.
 */

template <typename T> void fooRef(T& a)   { }
template <typename T> void fooVal(T  a)   { }
template <typename T> void fooPtr(T* arg) { }

int a [] = {1,2,3,4,5};
const int ca [] = {1,2,3,4,5};
int i = 10;
const int ci = 100;
int * pi = &i;
const int * pci = &i; // Pointer to Constant Int
const int * const cpci = &i; // Constant Pointer to Constant Int
int & ri = i;
const int & rci = ci; // Reference to Constant Int
```

Determine what type compiler chooses for parameter T. Choose "does not compile" if code is illegal?

A) `int`	
B) `const int`	
C) `int*`	5-1._____ `fooVal(ca);`
D) `int [5]`	5-2._____ `fooRef(a);`
E) `const int [5]`	5-3._____ `fooVal(ci);`
F) `does not compile`	5-4._____ `fooRef(ci);`
G) `const int *`	5-5._____ `fooVal(&ci);`
H) `int &`	5-6._____ `fooPtr(cpci);`
I) `const int &`	5-7._____ `fooVal<int&>(ci);`
J) `int * const`	
K) `const int * const`	

6. **Problem** (5 * 2 pts):

Given the definitions:

```
template <typename T> void foo(T a)       { cout << "1"; }
template <>           void foo(int  a)    { cout << "2"; }

template <typename T> void foo(T* a)      { cout << "3"; }
template <>           void foo(int* a)    { cout << "4"; }
template <>           void foo(double* a) { cout << "5"; }

                      void foo(int* a)    { cout << "6"; }

double d=1.0; int i=7; char ch='a';
```

What is printed for each of the following, choose "does not compile" if code is illegal?

A) 1	
B) does not compile	6-1._____ `foo(d);`
C) 4	6-2._____ `foo(&i);`
D) 5	6-3._____ `foo(&d);`
E) 2	6-4._____ `foo<int>(&i);`
F) 3	6-5._____ `foo<double>(d);`
G) 6	

7. **Problem** (6 * 2 pts):

Given the following definitions

```
template <typename T1, typename T2> bool compare(T1 lhs,T2 rhs)     {cout<<"1";}
template <>                         bool compare(int lhs, int rhs)  {cout<<"2";}

template <typename T1, typename T2> bool compare(T1 * lhs,T2 * rhs) {cout<<"3";}
template <> bool compare(const char * lhs, const char * rhs)        {cout<<"4";}
```

```
bool compare (int lhs, int rhs)                          {cout<<"5";}

const char * str1 = "A";
const char * str2 = "B";
double d1=1.0, d2=2.0;
int i1=1,i2=2;
```

what is printed in each of the following cases, choose NC if line does not compile:

A) 2 B) 5 C) 3 D) 1 E) 4 F) NC

7-1._____ `compare(i1,i2);`

7-2._____ `compare(str1,i2);`

7-3._____ `compare(&i1,&i2);`

7-4._____ `compare<int,int>(i1,i2);`

7-5._____ `compare<int,int>(d1,d2);`

7-6._____ `compare(str1,str2);`

8. **Problem** (3 * 2 pts):

Determine which of the following definitions are legal:

A) illegal

B) legal

8-1._____ `template <int x> int func() {return x;}`

8-2._____ `template <double x> double func() {return x;}`

8-3._____ `template <typename x> void func(x t) {}`

9. **Problem** (21 * 1 pts):

Given classes defined below, answer whether each of the following statements compiles or not:

```
template <typename T = int, int size = 10>
class Bar1 {
        T t;
    public:
        Bar1(const T& _t) : t(_t) { }
};

template <typename T = int, int size = 10>
class Bar2 {
        const T& t;
    public:
        Bar2(const T& _t) : t(_t) { }
};

template <typename T = int, typename T::U val = 11>
struct Bar3 { };

template <typename T = int, int val = T::four>
struct Bar4 { };

class A {
    private:
        A(const A&);
        A& operator=(const A&);
    public:
        A() : c_int2(200) {}
        A(int i) : c_int2(i) {}
```

```
        static int Get1() { return 15; }
        int Get2() const { return 15; }
        static const int c_int1 = 100;
        const int c_int2;
        typedef double U;
        enum Sizes { zero,one,two,three,four,five };
};

class B {
    public:
        typedef short U;
        enum Sizes { zero,one,two,three,four,five };
};
```

A) Compiles B) Does not compile

9-1._____ `Bar1<> b1(10);`

9-2._____ `Bar2<> b1_1(10);`

9-3._____ `Bar1<A> b3(10);`

9-4._____ `Bar2<A> b2(10);`

9-5._____ `Bar2<A,A::Get1()> b4(10);`

9-6._____ `Bar2<A,A::Get2()> b5(10);`

9-7._____ `Bar2<A,A::c_int1> b6(10);`

9-8._____ `Bar2<A,A::c_int2> b6_1(10);`

9-9._____ `Bar2<A,A::five> b7(10);`

9-10._____ `Bar2<A::U,A::five> b8(10);`

9-11._____ `Bar2<double,A::five> b9(10);`

9-12._____ `Bar2<double,A::Sizes::five> b10(10);`

9-13._____ `Bar3<> b11;`

9-14._____ `Bar3<double> b12;`

9-15._____ `Bar3<A> b13;`

9-16._____ `Bar3<B> b14;`

9-17._____ `Bar4<A> b15;`

9-18._____ `Bar4<B> b16;`

9-19._____ `Bar4<B,12> b17;`

9-20._____ `Bar4<B::U> b18;`

9-21._____ `Bar4<B::U,12> b19;`

10. **Problem** (6 * 2 pts):

Given class defined below, answer whether each of the following statements compiles or not. If it compiles, provide it's output:

```
template <typename T1, typename T2>
struct Bar { static const int value = 1; };

template <typename T1, typename T2>
struct Bar<T1*,T2> { static const int value = 2; };

template <typename T1, typename T2>
struct Bar<T1,T2*> { static const int value = 3; };

template <typename T1>
struct Bar<T1,T1*> { static const int value = 4; };
```

```
template <typename T2>
struct Bar<char,T2> { static const int value = 5; };

template <typename T2>
struct Bar<char*,T2> { static const int value = 6; };

template <>
struct Bar<char*,char> { static const int value = 7; };
```

A) Does not compile B) 2 C) 6 D) 3 E) 4 F) 5 G) 1 H) 7

10-1._____ `std::cout << Bar<int,int>::value << std::endl;`

10-2._____ `std::cout << Bar<int*,int>::value << std::endl;`

10-3._____ `std::cout << Bar<char,char>::value << std::endl;`

10-4._____ `std::cout << Bar<int,char*>::value << std::endl;`

10-5._____ `std::cout << Bar<char*,char*>::value << std::endl;`

10-6._____ `std::cout << Bar<char*,char>::value << std::endl;`

11. **Problem** (7 pts):

The following function

```
template <typename T1, typename T2>
void biggest( T1* a1, T2* a2 ) {
  T1 max1 = T1();
  T2 max2 = T2();
  for (int i=0; i<5; ++i) {
    if ( *a1 > max1 ) max1 = *a1;
    if ( *a2 > max2 ) max2 = *a2;
    ++a1;
    ++a2;
  }
}
```

DOES NOT compile with driver:

```
#include "biggest.h"
int main() {
  int a[] = {1,2,3,4,5};
  const int ca[] = {6,7,8,9,10};
  biggest(a,a);
  biggest(a,ca);
  biggest(ca,a);
  biggest(ca,ca);
}
```

the problem may be solved by overloading `biggest`. Points awarded for compactness. DO NOT change implementation, DO NOT use `cast`. USE GNU compiler.

Solution should be submitted through online script, use `assign_q2_biggest`. Filename `biggest.h`, I'll use driver as shown above.

12. **Problem** (7 pts):

The following function

```
template <typename T1, typename T2>
void func1( T1& a1, T2& a2 ) {
    T1 temp1;
    T2 temp2;
    temp1 = a1;
    temp2 = a2;
}
```

DOES NOT compile with driver:

```
#include "references.h"
int main() {
  int a = 1;
  const int ca = 6;
  func1(a,a);
  func1(a,ca);
  func1(ca,a);
  func1(ca,ca);
}
```

the problem may be solved by overloading `references`. Points awarded for compactness. DO NOT change implementation, DO NOT use `cast`. USE GNU compiler.

Solution should be submitted through online script, use `assign_q2_ref`. Filename `references.h`, I'll use driver as shown above.

13. **Problem** (15 pts):

Extend the following class

```
//forward declaration -- needed for friendship
//template <typename T> class Ptr;

template <typename T>
class Ptr {
  public:
    Ptr(T* _p) : p(_p) {}

    ~Ptr() { delete p; }
    T* Get() { return p; }

    //need friendship to access "p" in another instantiation
    //template <typename T2> friend class Ptr;
  private:
    T *p;
};
```

so that it works with the following driver

```
#include "templptr.h"
#include <iostream>

struct A { };

struct B {
  operator A () const { return A(); }
};

int main() {
  Ptr<int>    my_int_ptr1( new int (11) );
  Ptr<int>    my_int_ptr2( new int (22) );
  Ptr<int>    my_int_ptr3( my_int_ptr2 );

  std::cout
    << "int1 = " << *my_int_ptr1.Get() << " "
    << "int2 = " << *my_int_ptr2.Get() << " "
    << "copy = " << *my_int_ptr3.Get() << " "
    << std::endl;

  my_int_ptr1 = my_int_ptr2;

  std::cout
    << "assigned = " << *my_int_ptr1.Get() << " "
    << "int2 = " << *my_int_ptr2.Get() << " "
    << "copy = " << *my_int_ptr3.Get() << " "
    << std::endl;

  Ptr<float>  my_float_ptr1( new float (1.23f) );
  Ptr<float>  my_float_ptr2( new float (12.3f) );
  Ptr<double> my_double_ptr1( my_float_ptr1 );
  Ptr<double> my_double_ptr2( my_float_ptr2 );
  std::cout
    << "double1 = " << *my_double_ptr1.Get() << " "
    << "double2 = " << *my_double_ptr2.Get() << " "
    << std::endl;
  my_double_ptr2 = my_float_ptr1;
  std::cout
```

```
    << "double1 = " << *my_double_ptr1.Get() << " "
    << "double2 = " << *my_double_ptr2.Get() << " "
    << std::endl;

  Ptr<B> my_b_ptr ( new B() );
  Ptr<A> my_a_ptr ( my_b_ptr );
}
//EXPECTED OUTPUT
//int1 = 11 int2 = 22 copy = 22
//assigned = 22 int2 = 22 copy = 22
//double1 = 1.23 double2 = 12.3
//double1 = 1.23 double2 = 1.23
```

Make sure to test for memory leaks and memory errors. Points awarded for compactness. DO NOT change implementation of given methods, DO NOT use `cast`. USE GNU compiler.

Solution should be submitted through online script, use `assign_q2_templptr`. Filename `templptr.h`, I'll use driver as shown above.