quiz3. Code=1 Digipen login:________________________

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

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

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

   Determine which of the following definitions are legal:

   A) legal

   B) illegal

   3-1.______ `template <int x> int func() {return x;}`

   3-2.______ `template <double x> double func() {return x;}`

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

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

   This question deals with time-complexity of algorithms. Given `N` is the number of elements in a container, we have

   1) `O(N)` - algorithm runs in linear time, time proportional to the number of elements (slow)

   2) `O(log N)` - algorithm runs in logarithmic time, height of a balanced binary tree is `log(N)` where N is the number of elements in the tree (fast)

   3) `O(1)` - algorithm runs in constant time, that is the size of container does not matter, it'll take the same time for a container of `1, 100, 1000000, ...` elements (very fast)

   A) `O(N)` B) `does not compile` C) `O(log N)` D) `O(1)`

   4-1.______ `cont.push_back(val) in list`

   4-2.______ `cont.push_back(val) in set`

   4-3.______ `cont.insert(cont.begin(),10) in deque`

   4-4.______ `cont.push_front(val) in deque`

   4-5.______ `cont.find(val) in set (member find)`

5. **Problem** (15 * 1 pts):

   Let "cont" be an STL container. Answer whether the following lines compile for a given container type. Assume that container has more then 20 elements, and initial value of *iter* is a valid iterator.

   A) compiles B) does not compile

   5-1.______ `int i = cont[10]; //vector<int>`

   5-2.______ `int i = cont[10]; //list<int>`

   5-3.______ `int i = cont[10]; //set<int>`

   5-4.______ `cont.insert(10); //vector<int>`

   5-5.______ `cont.insert(10); //list<int>`

   5-6.______ `cont.insert(10); //set<int>`

   5-7.______ `iter=cont.begin(); iter++; //vector<int>`

   5-8.______ `iter=cont.begin(); iter++; //list<int>`

   5-9.______ `iter=cont.begin(); iter++; //set<int>`

   5-10.______ `iter=cont.begin(); iter+5; //vector<int>`

   5-11.______ `iter=cont.begin(); iter+5; //list<int>`

   5-12.______ `iter=cont.begin(); iter+5; //set<int>`

   5-13.______ `iter=cont.begin(); *iter=5; //vector<int>`

   5-14.______ `iter=cont.begin(); *iter=5; //list<int>`

   5-15.______ `iter=cont.begin(); *iter=5; //set<int>`

6. **Problem** (8 pts):

   There is a memory leak - **fix** it. **Modify main only.**

```
#include <vector>
class Foo {
  int * pi;
  public:
  Foo(int i) : pi( new int (i) ) {}
  Foo(const Foo& rhs) : pi( new int (*(rhs.pi)) ) { }
  ~Foo() { delete pi; }
};

int main () {
  std::vector<Foo*> cont;
  for (int i=0; i<10; ++i) cont.insert( cont.end(), new Foo(i) );
}
```

   Your code should be in a single file, say `stl-leak.cpp`, which you should submit online by copying/pasting your code. For grading I'll use driver as shown above. Grading will be done with GNU compiler.

7. **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.

Your code should be in a single file, say `templptr.h`, which you should submit online by copying/pasting your code. For grading I'll use driver as shown above. Grading will be done with GNU compiler.

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

   In this question you will implement a member function adapter for a member function that takes a single argument. Refer to the example implementation of member function adapter with no arguments, see class web-site.

```
#include <iostream>
#include <vector>
#include <list>
#include <algorithm>
#include "mem.fun.with.arg.h"

class A {
  int i;
  public:
    A(int _i) : i(_i) {}
    const int& Get() const { return i; }
};

class C {
  int i;
  public:
    C(int _i) : i(_i) {}
    A Multiply(const int& arg2) { return A(i*arg2); }
    const int& Get() const { return i; }
};

int main () {
  std::vector<C> v;
  for (int i=0;i<10; ++i) v.push_back( C(i) );
  std::vector<C>::const_iterator it=v.begin(), it_end=v.end();
  for ( ; it!=it_end; ++it) {
    std::cout << "C(" << it->Get() << ") ";
  }
  std::cout << std::endl;

  //second input range for transform
  std::list<int> list;
  for (int i=0;i<10; ++i) list.push_back( 10-i );
  //soutput range for transform (filled with garbage)
  std::vector<A> out;
  for (int i=0;i<10; ++i) out.push_back( A(0) );

  //=====================================
  std::transform(
      v.begin(), v.end(), list.begin(), out.begin(), mfp_1arg( &C::Multiply ) );
  for ( std::vector<A>::const_iterator itA=out.begin(); itA!=out.end(); ++itA) {
    std::cout << "A(" << itA->Get() << ") ";
  }
  std::cout << std::endl;
}
//expected output
//C(0) C(1) C(2) C(3) C(4) C(5) C(6) C(7) C(8) C(9)
//A(0) A(9) A(16) A(21) A(24) A(25) A(24) A(21) A(16) A(9)
```

Your code should be in a single file, say `mem.fun.with.arg.h`, which you should submit online by copying/pasting your code. For grading I'll use driver as shown above. Grading will be done with GNU compiler.

9. **Problem** (10 pts):

   Implement an overload of transform that takes inputs from 3 ranges and outputs the result into a fourth, so that `main.cpp` below compiles/runs and has no memory leaks/errors.

   Points awarded for correctness and compactness. DO NOT change implementation of given methods, DO NOT use `cast`.

   Your code should be in a single file, say `transform3.h`, which you should submit online by copying/pasting your code. For grading I'll use driver as shown below. Grading will be done with GNU compiler.

```
#include <iostream>
#include <vector>
#include <list>
#include <set>
#include <deque>
#include "transform.h"

int sumof3(const int&i1, const int&i2, const int&i3 ) {
  return i1+i2+i3;
}
```

```
class Counting {
  public:
    Counting() : total(0) {}
    int operator() (const int&i1, const int&i2, const int&i3 ) {
      int delta = i1+i2+i3;
      total += delta;
      return delta;
    }
    int GetTotal() const {
      return total;
    }
  private:
    int total;
};

int main () {
  std::vector<int> v;
  std::list<int>   l;
  std::set<int>    s;
  std::deque<int>  d;
  for (int i=0;i<10; ++i) {
    v.push_back( i );
    l.push_back( 20-2*i );
    s.insert( i );
    d.push_back( 0 );
  }

  transform(v.begin(), v.end(), l.begin(), s.begin(), d.begin(), sumof3 );

  for ( std::deque<int>::const_iterator it=d.begin(); it!=d.end(); ++it) {
    std::cout << *it << " ";
  }
  std::cout << std::endl;

  Counting c =
    transform(v.begin(), v.end(), l.begin(), s.begin(), d.begin(), Counting() );

  for ( std::deque<int>::const_iterator it=d.begin(); it!=d.end(); ++it) {
    std::cout << *it << " ";
  }
  std::cout << std::endl;
  std::cout << "Total =  " << c.GetTotal() << std::endl;
}
//expected output
//20 20 20 20 20 20 20 20 20 20
//20 20 20 20 20 20 20 20 20 20
//Total =  200
```