final-sample. Code=1 Digipen login:_____

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

   PUT YOUR DIGIPEN LOGIN ON BOTH SIDES OF EACH SHEET OF THE EXAM – use upper-right corner

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

   Use this part of the exam for scratch work. When done:

   (a) copy multiple choice answers to answer sheet – provided separately, submit your choices (characters A,...,Z – not the actual answers) online no later then midnight of the exam day. Don't forget to copy your testid.

   (b) all written questions should be answered on a yet another answer sheet – provided separately.

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._____ `b.f1();` 3-4._____ `pb2->f2();` 3-5._____ `b.f3();` 3-6._____ `pb2->f1();`

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

Given these classes

```
class B {
  public: virtual std::string name() { return "B"; }
};

class D : public B {
  public: virtual std::string name() { return "D"; }
};
```

what is the output of each of the following `main`s? Notice that each `main` uses a different function `foo`.

A) In foo: D In main: B

B) In foo: B In main: B

C) In foo: D In main: D

D) In foo: B In main: D

4-1._____

```
B foo(B& b) { std::cout << "In foo: " << b.name(); return b; }

int main() {
  D d;
  std::cout << "In main: " << foo(d).name();
}
```

4-2._____

```
B foo(B b) { std::cout << "In foo: " << b.name(); return b; }

int main() {
  D d;
  std::cout << "In main: " << foo(d).name();
}
```

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

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 C) 4 D) 5 E) 2 F) 3 G) 6	6-1._____ `foo(d);` 6-2._____ `foo(&i);` 6-3._____ `foo(&d);` 6-4._____ `foo<int>(&i);` 6-5._____ `foo<double>(d);`

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

What is the output of the program shown below for each of the following values of "???"

```
void foo(int val) {
  int i = 5;
  double d = 15.5;
  std::cout << "1";       //PRINT STATEMENT
  switch (val) {
    case 1: //throw int
      throw i;
      break;
    case 2: //throw double
      throw d;
      break;
    case 3: //throw nothing
      break;
  }
  std::cout << "2";       //PRINT STATEMENT
}

int main() {
  std::cout << "3";       //PRINT STATEMENT
  try {
    foo(???); // ??? is substituted by a number - see below
  }
  catch (int ex) {
    std::cout << "4";     //PRINT STATEMENT
  }
  std::cout << "5";        //PRINT STATEMENT
}
```

A) 3125 B) 31245 C) 3124 D) 314 E) 312 F) 31 G) 315 H) 3145

7-1._____ if "???" is substituted by 1

7-2._____ if "???" is substituted by 2

7-3._____ if "???" is substituted by 3

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

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

```
template <typename T1 =
                int T2 =
class Bar {
  public:
    Bar(int x = 0) { }
  private:
    T1 items[T2];
};
```

```
int,
10>
 class A {
   public:
     A() { }
};
```

```
class B {
  public:
    B(int x) : x_(x) { }
    operator int(void) {
      return x_;
    }
  private:
    int x_;
};
```

A) Does not compile B) Compiles

8-1._____ `Bar<int, 5> bar1;`

8-2._____ `Bar bar2(5);`

8-3._____ `Bar<B, 5> bar5;`

8-4._____ `Bar<> bar8;`

8-5._____ `Bar<5> bar9;`

8-6._____ `int size = 8; Bar<int, size> bar11;`

9. **Problem** (4 pts):

When must template *class* have explicit template parameters?

A) Never, the template types can always be inferred

B) Always

C) When the template types cannot be inferred

_____

10. **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 "iter" is an iterator corresponding to the container.

A) does not compile B) compiles

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

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

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

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

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

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

10-7._____ `iter=cont.begin(); iter++; //vector<int>`

10-8._____ `iter=cont.begin(); iter++; //list<int>`

10-9._____ `iter=cont.begin(); iter++; //set<int>`

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

10-11._____ `iter=cont.begin(); iter+5; //list<int>`

10-12._____ `iter=cont.begin(); iter+5; //set<int>`

10-13._____ `iter=cont.begin(); *iter=5; //vector<int>`

10-14._____ `iter=cont.begin(); *iter=5; //list<int>`

10-15._____ `iter=cont.begin(); *iter=5; //set<int>`

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

```
class B {
  public:
    B() {}
    ~B() {}
};

class D : public B {
    int * pi;
```

```
  public:
    D(int i) : pi( new (i) ) {}
    ~D() { delete pi; }
    //other methods: copy, assign, ctor are defined here
    //do not implement  - they are not relevant to the problem
};

int main() {
  B* pd = new D(100);
  delete pd;
}
```

Are there any problems with the above code at compile-time or run-time? Identify the problems and fix the code – **do not modify** `main`.

12. **Problem** (4 pts):

```
class B {
  public:
    B(int _i);
    B& operator=(const B& rhs) { i=rhs.i; }
  private:
    int i;
};

class D : public B {
  public:
    D& operator=(const D& rhs);
};
```

Implement derived assignment operator – **do not modify class B**. Notice that D cannot access `B::i` directly, since the latter is private.

13. **Problem** (5 pts):

What is the difference between single element `delete` and array `delete []`?

Calculate memory leak size for the following program (show work):

```
class C {
  int * array;
  public:
  C() : array( new int [10] ) {}
  ~C() { delete [] array; }
};

int main() {
  C* array_of_C = new C[10];
  delete array_of_C; // ERROR !!!
}
```

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

```
#include <iostream>
class Vector3 {
  public:
    Vector3() : v(new int[3]) {
      for (unsigned i=0;i<3; ++i) v[i]=0;
    }
    //appropriate copy ctor and assignment operator
    //do not implement - they are not relevant to this problem
    ~Vector3() { delete [] v; }
    int& operator[] (const int & index) {
      return v[index];
    }
  private:
    int * v;
};
```

Which lines of this main DO NOT COMPILE?

```
int main() {
  { Vector3 v;       int i = v[1]; } //line 1
  { const Vector3 v; int i = v[1]; } //line 2
  { Vector3 v;       v[1]  = 5; }    //line 3
  { const Vector3 v; v[1]  = 5; }    //line 4
}
```

Which of the 4 lines of main compile?

From the client's point of view - which line(s) of the main SHOULD compile and which SHOULD NOT?

Modify `operator[]` (add new methods if needed) so that `Vector3` works correctly from the client's point of view.

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

```
class C; //forward declaration

class Cpointer {



  public:









};

class C { //do not modify
    int i;
  public:
    C(int _i=0) : i(_i) {}
    Cpointer operator& () { return Cpointer(this); }
    void Do() { std::cout << "i=" << i << std::endl; }
};

int main() {
  Cpointer p (new C(100));
  p->Do();
}
```

Complete class `Cpointer` so that the code compiles. Make sure there are no memory leaks.

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

The standard `copy` algorithm copies a range of elements from a source range into a destination. However, there is no standard `copy_if` algorithm. A `copy_if` algorithm performs the copy when the specified predicate returns *true*.

Implement a templated `copy_if` algorithm. (Hint: The function takes 4 parameters.)

```
template <

            copy_if(
```

What is your return value?

Which category of iterators is/are required in your implementation?

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

In this question you are required to implement something similar to `std::bind2nd`. For simplicity main is written using a "for"-loop, but it may be rewritten with "for_each" and STL containers.

```
class Functor1 {
  public:
    int operator()(const int & i, const int & j) const {
      return i+j;
    }
};

class Functor2 {
  public:
    int operator()(const int & i, const int & j) const {
      return i*j;
    }
};

template <typename T>
class BindSecArg









};

int main () {
  Functor1 f1;
  for (int i=0; i<10; ++i) std::cout << f1(i,i) << " "; //0 2 4 6 8 10
  std::cout << std::endl;

  Functor2 f2;
  for (int i=0; i<10; ++i) std::cout << f2(i,i) << " "; //0 1 4 9 16 25
  std::cout << std::endl;

  BindSecArg<Functor1> b1(4); //bind second argument of Functor1 to 4
  for (int i=0; i<10; ++i) std::cout << b1(i) << " "; //4 5 6 7 8 9
  std::cout << std::endl;

  BindSecArg<Functor2> b2(4); //bind second argument of Functor2 to 4
  for (int i=0; i<10; ++i) std::cout << b2(i) << " "; //0 4 8 12 16 20
  std::cout << std::endl;
}
```

Extra credit question: your implementation most probably doesn't work (which is OK!) with

```
class Functor3 {
  public:
    std::string operator()(const std::string & i, const std::string & j) const {
      return i+j;
    }
};
```

how does STL solve this problem?

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

Declare and implement a functor `F` whose `operator()` takes 2 arguments and returns the first to the power of second ($(b, p) \rightarrow b^p$). Assume argument types are compatible with the operations that you use in implementation. DO NOT provide default values to any of the `operator()` arguments. Read the next 2 questions in this problem to get a better idea on the functionality of this functor.

Write one line of code using STL algorithm and the functor `F` that for given 2 integer containers `bases` and `powers` produces another container `results` by raising each element of the first container to the power the corresponding element from the second container. Example:

```
2   2  3  4   5   <- bases
5   4  3  2   2   <- powers
32 16 27 16  25   <- results
```

Write one line of code using the functor `F` that squares each element of an integer container `bases` and writes the result back to the same container. Example:

```
1  2  3  4   5   <- bases before
1  4  9 16  25   <- bases after
```