final. Code=3 Digipen login:__________________________

1. **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. Do not forget to write down your testid.

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

2. **Problem** (6 * 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        MemGetInt()         const { return member; }
    int&       MemGetRefInt()      const { return member; }
    const int& MemGetRefConstInt() const { return member; }

  private:
    int * data;
    int   member;
};
```

A) compiles B) fails to compile	2-1.______ GetInt() 2-2.______ GetRefInt() 2-3.______ GetRefConstInt() 2-4.______ MemGetInt() 2-5.______ MemGetRefInt() 2-6.______ MemGetRefConstInt()

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

   Given the following definitions

```
class B1 {
      int i;
    public:
        B1() : i(1) {}
        virtual int foo() { return i; }
};
class D1 : public B1 {
      int j;
    public:
        D1() : j(2) {}
        int foo() { return j; }
};

class B2 {
      int i;
    public:
        B2() : i(3) {}
        int foo() { return i; }
};
class D2 : public B2 {
      int j;
```

```
    public:
        D2() : j(4) {}
        int foo() { return j; }
};

int f1() {
    B1* p = new D1;
    return p->foo();
}

int f2() {
    B2* p = new D2;
    return p->foo();
}

int f3() {
    B1* p = new B1;
    return static_cast<D1*>(p)->foo();
}

int f4() {
    B2* p = new B2;
    return static_cast<D2*>(p)->foo();
}

int f5() {
    B2* p = new D2;
    return dynamic_cast<D2*>(p)->foo();
}

int f6() {
    B1* p = new B1;
    if ( D1* pd = dynamic_cast<D1*>(p) ) return pd->foo(); else return 0;
}
```

What happens in each of the following function calls (each function is compiled and ran separately):

A) compiles, but run-time error B) compiles and returns 3 C) compiles and returns 1 D) does not compile E) compiles and returns 0 F) compiles and returns 4 G) compiles and returns 2	3-1.______ `foo1();` 3-2.______ `foo2();` 3-3.______ `foo3();` 3-4.______ `foo4();` 3-5.______ `foo5();` 3-6.______ `foo6();`

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

Given the following definitions

```
class B {
  public:
    void f1();
  protected:
    void f2();
  private:
    void f3();
};

class D : public B {
  public:
    void f4();
};

class C {
  public:
    void f5();
  private:
    B b;
};

D d;
C c;
```

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

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

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.
 */
For each of the following template function calls
determine which value of the template parameter T the compiler will use?

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

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

6. **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?

   A) 5 B) 4 C) 1 D) does not compile E) 3 F) 2

   6-1.______ `compare(i1,i2);`

   6-2.______ `compare(str1,i2);`

   6-3.______ `compare(&i1,&i2);`

   6-4.______ `compare<int,int>(i1,i2);`

   6-5.______ `compare<int,int>(d1,d2);`

   6-6.______ `compare(str1,str2);`

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

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

```
template <typename T1 = int,
                int T2 = 10>
class Bar {
  public:
    Bar(int x = 0) { }
  private:
    T1 Do(T1 arg) {
      return arg;
    }
};
```

```
class A {
  public:
    A() { }
  private:
    A(const A&) {} //PRIVATE COPY
};
```

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

   A) Compiles B) Does not compile

   7-1.______ `Bar<int, 5> bar1;`

   7-2.______ `Bar<int, B(5)> bar3;`

   7-3.______ `Bar<A> bar4(B(5));`

   7-4.______ `Bar<B, 5> bar6(5);`

   7-5.______ `Bar<> bar8;`

   7-6.______ `Bar<5> bar9;`

   7-7.______ `int size = 8; Bar<int, size> bar11;`

   7-8.______ `Bar<B(5)> bar12;`

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

```
class B {
  public:
    B(int _i);
  private:
    int i;
};

class D : public B {
  public:
    D(int _i,int _j); // _i to initialize B::i, _j for D::j
  private:
    int j;
};
```

Implement derived constructor. Notice that D cannot access B::i, since the latter is private.

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

   What does this program output?

```
class B {
  public:
    int b;
    B(int _b=0) {
        b=_b;
        std::cout << "B("<< b <<")" << std::endl;
    }
    // not virtual !!!
    ~B() { std::cout << "~B("<< b <<")" << std::endl; }
};

class D : public B {
  public:
    int d;
    D(int _b=0, int _d=0) {
      b=_b;
      d=_d;
      std::cout << "D("<< _b <<","<< d <<")" << std::endl;
    }
    ~D() { std::cout << "~D("<< b <<","<< d <<")" << std::endl; }
};

int main() {
  B** arr = new B*[3];
  arr[0] = new B(1);
  arr[1] = new D(2,5);
  arr[2] = new D(3,3);

  std::cout << "------\n";

  for (int i=0;i<3;++i) delete arr[i];

  delete [] arr;
}
```

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

```
class C {
  private:
    int i;
  public:
    C()              : i(0)     { std::cout << "C()\n"; }
    C(int _i)        : i(_i)    { std::cout << "C(int)\n"; }
    C(const C& rhs) : i(rhs.i) { std::cout << "C(const C&)\n"; }

    C& operator=(const C& rhs) {
      std::cout << "operator=(const C&)\n";
      i = rhs.i;
      return *this;
    }

    C operator+ ( C const& arg2) const {
      std::cout << "operator+(C const&)\n";
      C ret_val( *this );
      ret_val.i += arg2.i;
      return ret_val;
    }

    ~C() { std::cout << "~C()\n"; }
};

int main() {
  C c1;
  C c2(12);
  C c3(c1);
  std::cout << "-----" << std::endl;
  c1 = c2 + 10;
  std::cout << "-----" << std::endl;
}
```

(a) what is the output of this program?

(b) rewrite `operator+` to make use of return value optimization.

(c) what is the output with your implementation of `operator+`

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

Write a complete program to demonstrate the use of arrays of base class pointers. The program should have:

- classes Animal, Dog, and Cat
- method `void Talk()`

in main:

- create an array of "Animal*"
- manually populate it (i.e. you do not have to use "for"-loop)
- call Talk on all pointers (substitute dots with appropriate code)
  `for (int i=0;i<5;++i) ....[i].....Talk();`
- do stuff that you think is appropriate

the output of the program should be (white space will be ignored - so it's OK if you use new-lines):

`meawu woof woof meawu meawu`

Points awarded for correctness, compactness, completeness, no memory leaks.

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

```
template <typename T>
class SmartPointer {
    //add code here
  public:
    //add code here
};

class C { //do not modify this class
    int i;
  public:
    C(int _i=0) : i(_i) {}
    operator int () const { return i; }
    void Set(int _i) { i=_i; }
};

int main() { //do not modify this function
  SmartPointer<C> p (new C(100));
  int i = *p; //i should be 100
}
```

(a) Complete class `SmartPointer` so that `main` compiles. Make sure there are no memory leaks.

(b) What functionality should be added to `SmartPointer` to support code like this (explain – DO NOT IMPLEMENT)

```
SmartPointer<C> p3(new C(100));
{
  SmartPointer<C> p4(p3);
  p4->Set(22);
}
int i = *p3; //should be 22
```

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

    The standard `remove_copy_if` algorithm copies a range of elements from a source range into a destination, except that elements for which predicate is *true* are not copied.

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

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

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

Implement a program (`main` function) which

- allocates an STL vector of 10 pointers to objects of type Foo
- uses `for_each` and a user-defined functor (you have to provide it's implementation) to call `Print` on each of the elements of the vector

Make sure your code has no memory leaks.

Extra credit: sort the vector with respect to the integer pointed to by *pi*. Use `std::sort`.

```
class Foo {
    int * pi;
  public:
    Foo() : pi( new int ( std::rand() % 20 ) ) {} //random integer 0....19
    Foo(const Foo& rhs) : pi( new int (*(rhs.pi)) ) { }
    void Print()  const { std::cout << "Foo(" << *pi << ")\n"; }
    int  GetVal() const { return *pi; }
    ~Foo()              { delete pi; }
};

int main () {
```

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

    As we have seen STL provides capabilities of applying operations on ranges using function objects (functors). For example to add corresponding elements from 2 containers and place the result into a third container:

```
  std::transform(list.begin(),list.end(),
                 vec.begin(),
                 result.begin(),
                 std::plus<int>());
```

    Mathematically `std::plus<int>()` is a function $f(x, y)$, which is applied to x from the first container (list), y from the second (vec). One may need the following functionality: instead of passing unprocessed x and y to function f, first apply g: $f(g(x), g(y))$. Which is usually referred to as function composition.

    Implement composition as a functor `Compose` so that code below compiles and runs.

```
class Square {
  public: int operator() (const int & arg) const { return arg*arg; }
};

class Cube {
  public: int operator() (const int & arg) const { return arg*arg*arg; }
};

class Compose {

};

int main () {
  std::list<int> list;
  std::vector<int> vec;
  //assume containers are initialized to
  //list 7 5 3 1
  //vec  1 2 3 4
  std::vector<int> result(4); //set size to 4

  //apply composition
  std::transform(list.begin(),list.end(),
                 vec.begin(),
                 result.begin(),
                 Compose< std::plus<int>, Square> () );

//result now is 50 29 18 17
//where values are 50=7^2+1^2, 29=5^2+2^2, 18=3^2+3^2, 17=1^2+4^2

  std::transform(list.begin(),list.end(),
                 vec.begin(),
                 result.begin(),
                 Compose< std::plus<int>, Cube> () );

//result now is 344 133 54 65
//where values are 344=7^3+1^3, 133=5^3+2^3, 54=3^3+3^3, 65=1^3+4^3
}
```

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

Given the following 2 functors:

```
struct IsSquare {
    bool operator() ( int const& el ) const { // true if argument IS a square
        int root = std::sqrt( el );
        return ( el == root*root );
    }
};

struct IsEven {
    bool operator() ( int const& el ) const { // true is argument IS even
        return ( el % 2 == 0 );
    }
};
```

make the following function to compile and run. Expected output of print is provided in comments.

`not1` is an adaptor that negates the result of the corresponding functor.

```
template <typename Container>
void print ( Container const& cont ) {
    typename Container::const_iterator it = cont.begin(), it_end = cont.end();
    while(  it != it_end ) { std::cout << *it << " "; ++it; }
    std::cout << std::endl;
}

int main() {
    std::vector<int> v;
    v.push_back(1); v.push_back(2); v.push_back(3); v.push_back(4); v.push_back(5);
    v.push_back(6); v.push_back(7); v.push_back(8); v.push_back(9); v.push_back(10);
    print( v ); // 1 2 3 4 5 6 7 8 9 10

    //remove all elements that ARE NOT squares
    std::vector<int>::iterator end_new = std::remove_if( v.begin(), v.end(), not1( IsSquare() ) );
    v.erase( end_new, v.end() );
    print( v ); // 1 4 9

    //remove all elements that ARE NOT even
    end_new = std::remove_if( v.begin(), v.end(), not1( IsEven() ) );
    v.erase( end_new, v.end() );
    print( v ); // 4
}
```