University of Toronto

Duration: 50 minutes
Examiner: Kuei (Jack) Sun

Please fill your student number, last and first name below and then read the instructions carefully.

Student Number: ____ ____ ____ ____ ____ ____ ____ ____ ____ ____

Last Name: _____

First Name: _____

Instructions

Examination Aids: Ruler and examiner provided aid sheet are allowed.

Do not turn this page until you have received the signal to start.

You may remove the aid sheet from the back of this test book. Do not remove any other sheets from this test book. Answer all questions in the space provided. No additional sheets are permitted.

This exam consists of 5 questions on 8 pages (including this page). The value of each part of each question is indicated. The total value of all questions is 70 marks.

For the written answers, explain your reasoning clearly. Be as brief and specific as possible. Clear, concise answers will be given higher marks than vague, wordy answers. Marks will be deducted for incorrect statements in an answer. Please write legibly!

Work independently.

MARKING GUIDE

Q1:	_____	(10)
Q2:	_____	(8)
Q3:	_____	(17)
Q4:	_____	(15)
Q5:	_____	(20)
Total:	_____	(70)

Question 1. **True or False** [10 marks]

Circle **T** if the statement is true, otherwise circle **F** if the statement is false. 2 marks each.

1. C++ only allows default argument for primitive types. **T** **F**

2. In Python, immutability is a more restrictive form of hashability. **T** **F**

3. Python decorator always returns a callable object. **T** **F**

4. It is possible for C3 linearization to fail while creating method resolution order. **T** **F**

5. C macros are designed to prevent infinite recursion during macro expansion. **T** **F**

Question 2. **Multiple Choices** [8 marks]

Pick all answer(s) that are correct. You will lose 1 mark per wrong choice, down to 0 marks.

a) Which of the following special Python methods are used by descriptors? [4 marks]

i. __get__

ii. __getattr__

iii. __getattribute__

iv. __del__

v. __delete__

b) Which of the following language properties apply to Python? [4 marks]

i. Strongly typed (with respect to use of coercion)

ii. Type-safe

iii. Manifest typing

iv. Duck typing

v. High-level

Question 3. Short Questions [17 marks]

a) Use dictionary comprehension to create a histogram of a list of immutable objects. [5 marks]

```
a = [1, 2, 5, "hi", 2, "hi", "bye", 2, 9 ]
# would create {1: 1, 2: 3, 5: 1, "hi": 2, "bye": 1, 9: 1}
```

```
{ x : a.count(x) for x in set(a) }
```

2 marks for a.count(x)

1 mark for the loop (you actually don't need set(a). "for x in a" works too)

1 mark for correct dictionary comprehension notation (k : v)

1 mark for enclosing with braces (not with brackets!)

b) Explain the *semantic* difference between the assignment operators in Python versus C++. Hint: statement vs. expression is not the answer. [4 marks]

In Python, assignment is always by reference. It performs name binding. (2 marks)

In C++, assignment is always by copy or move (except when you initialize a reference variable).

(2 marks)

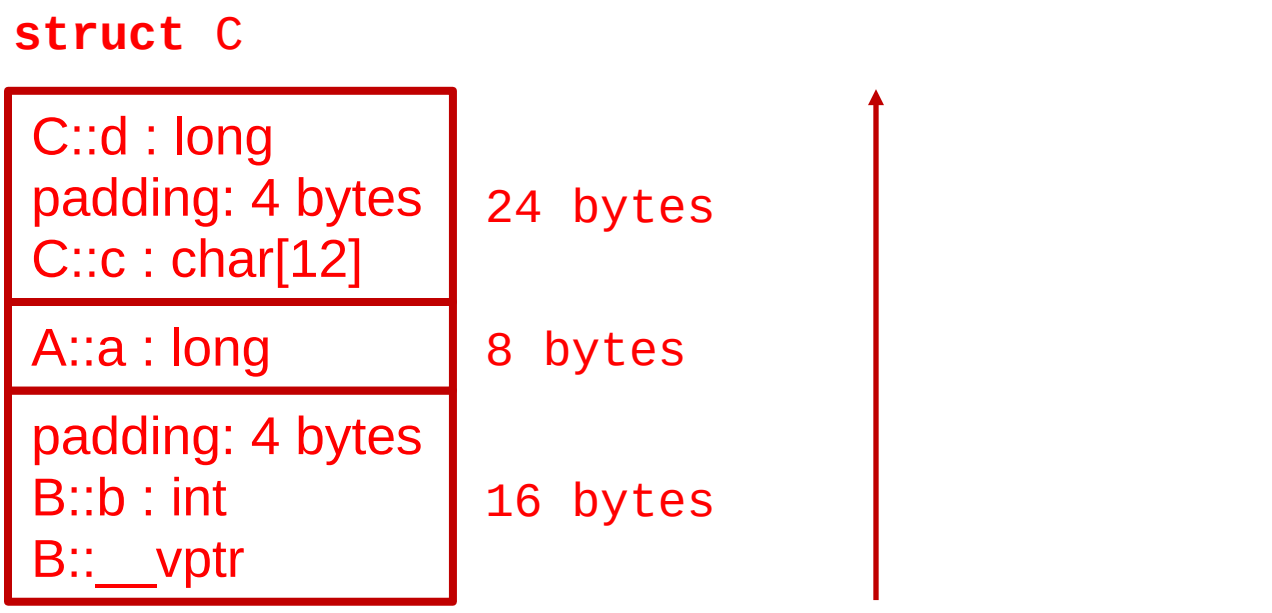
c) Draw the data layout for the derived class C, and state the size of C. Assume 8 byte alignment. Indicate whether higher address goes up or down. [8 marks]

```
struct A {
  long a;
};

struct B {
  int b;
  virtual void foo() { cout << "B::foo" << endl; }
};

struct C : public B, public A {
  char c[12];
  long d;
  virtual void foo() { cout << "C::foo" << endl; }
};
```

1 mark for each variable and padding in the structure. -1 mark if structure is out of order. -1 mark if the fields within a structure is out of order. 1 mark for calculating the final size correctly.



sizeof(C) = 48

Question 4. Python Metaclass [15 marks]

Write a Python metaclass, `Dict`, such that the class it creates works like a dictionary. Hint, you will need to overload the __getitem__ and __setitem__ operators. Assume __str__ is already implemented. Note: you may need to customize attribute access.

```
class Foo(metaclass=Dict):
    pass

class Dict(type):
```

```
>> Foo["hello"] = 3
>> Foo.update({
..  "bye": 2,
..  "world" : 4,
.. })
>> print(Foo)
{'hello' : 3, 'bye': 2,
 'world': 4}
```

```
    def __init__(cls, name, bases, attrs):
        cls._dict = {}

    def __setitem__(cls, key, value):
        cls._dict[key] = value

    def __getitem__(cls, key):
        return cls._dict[key]

    def __getattr__(cls, name):
        return getattr(cls._dict, name)


5 marks for __init__
3 marks for __setitem__ and __getitem__
4 marks for __getattr__

Alternative implementation

class Dict(type):
     _mapping = {}
     def __init__(cls, name, bases, attrs):
          cls._mapping[cls] = {}
          # or Dict._mapping[cls] = {}

     def __getitem__(cls, key):
        return cls._mapping[cls][key]

     …
```

Question 5. C++ Template Programming [20 marks]

Given a generic binary search tree implementation shown here:

```
typedef int (*cmp_f)(const void * const lhs, const void * const rhs);
typedef void (*dst_f)(void * p);

class GenericBST {
     struct Node {
       void * data;
          Node * left, * right;
          Node(void * d) : data(d), left(nullptr), right(nullptr) {}
     } * root;
     cmp_f compfunc;
     dst_f destfunc;
     void destroy(Node * nd) {
          if (nd) {
               destroy(nd->left);     destroy(nd->right);
               destfunc(nd->data);    delete nd;
          }
     }
public:
     GenericBST(cmp_f cf, dst_f df) : root(nullptr), compfunc(cf),
          destfunc(df) {}
     ~GenericBST() { destroy(root); }

  /* if key exists, returns node->data, else nullptr */
     const void * lookup(const void * const key);

  /* return false if key exists, else true for successful insert */
     bool insert(void * key);

  /* return false if key not found, true for successful deletion */
     bool remove(const void * const key);
};

void destroy_int(void * ptr) { delete (int *)ptr; }

/* returns positive if greater, negative if lesser, 0 otherwise */
int compare_int(const void * const lhs, const void * const rhs) {
     return *(int *)lhs - *(int *)rhs;
}
GenericBST ibst(compare_int, destroy_int);
```

Write a template that is a subclass to GenericBST to make it specific to parameterized type T. You may assume the type T supports all comparison operators. Your template class must turn all generic interfaces into type-specific ones (e.g, you need to implement lookup, insert, remove, and the constructor). Note that insert *does not* allocate, it just stores the pointer given to from user. An example use is shown here:

```
int a = new int(5);
if (ibst.insert(a) == false)
     delete a;
```

Your template class should make it so that its insert function will make a copy prior to insertion, so that the user of your template class does not need to manually delete upon insertion failure.

```
template<typename T>
class BST : private GenericBST {
```

```
2 marks for lookup
2 marks for remove
6 marks for compare (can be different name)
5 marks for insert
2 marks for destroy (can be different name)
2 marks for constructor
1 mark for public keyword placed correctly
```

(jack): Note that I changed the solution to use composition instead of private inheritance. They are equivalent in this case because we have to create wrappers for all methods provided by GenericBST.

```
template<typename T>
class BST {
     GenericBST gbst;

     static int compare(const void * const lhs,
                        const void * const rhs)
     {
          const T * left = (const T *)lhs;
          const T * right = (const T *)rhs;

          if (*left < *right)
               return -1;
          else if (*left > *right)
               return 1;
          return 0;
     }

     static void destroy(void * ptr) {
          delete (T *)ptr;
     }
public:
     BST() : gbst(&compare, &destroy) {}

     const T * lookup(const T & key) {
          return (const T *)gbst.lookup(&key);
     }

     bool insert(const T & key) {
          if (gbst.lookup(&key) != nullptr)
               return false;
          T * copy = new T(key);
          return gbst.insert(copy);
     }

     bool remove(const T & key) {
          return gbst.remove(&key);
     }
};
```

END OF EXAMINATION