CMPS 109: Advanced Programming

Mid-Term Examination
Group (A)
Fall 2016
Thursday 27th October, 2016
University of California Santa Cruz

Student Name: ____

Cruz Id: ____________________@ucsc.edu

Grades:

Question	Total Grade	Student Grade
Q1	10	
Q2	10	
Q3	10	
Q4	15	
Q5	15	
Q6	10	
Q7	10	
Q8	20	
Total	100	

Q1. Identify false and true statements and justify your answer or provide a correction for the false statements. Draw a Circle around T or F.

1. Inline functions are guaranteed to be inlined. (T) [✓ (F)]

2. A pure virtual method is used to guarantee/enforce its implementation in the base and the descendant classes. [✓ (T)] (F)

3. There is no solution for the inheritance diamond problem. (T) [✓ (F)]

4. Static functions are globally accessible while static variables are local to their libraries scope. (T) [✓ (F)]

5. Private inheritance entails access to the private methods of the base class by the descendant classes. (T) (F)

Q2. Choose all the correct answers. Identify your answer by a circle around your selected choices:

1.To enforce inline functions to be inlined:
- a. A function should be defined as static.
- [✓ b. Use the always_inline attribute.]
- c. Use compiler optimization switches.
- [✓ d. Define the function as static and use compiler optimization switches.]

2. The output of the preprocessing phase is:
- a. Optimized source code.
- [✓ b. Source code after preprocessing macros and includes.]
- c. Assembly code.
- d. None of the above.

3. Class templates are processed during:
- a. Preprocessing time.
- [✓ b. Compile time.]
- c. Machine code generation time.
- d. Link time.

4. To disallow descendant classes from extending specific methods, they should be defined as:
- a. Virtual methods.
- b. Pure virtual methods.
- c. Explicit methods.
- [✓ d. None of the above.]

5. To disable implicit type conversion of classes:
- a. Use compiler command line switches.
- b. Define the class constructor as final.
- c. Define the class constructor as explicit.
- d. Any of the above.

Q3. Does the following program compile successfully. If yes, show the output or else explain why.

```
#include <iostream>
using namespace std;

class A{
    static int nextid;
    int id, val;
  public:

    void print(string s, ostream& o=std::cout) const {
      o << "A::" << s << " id== " << id
        << " val== " << val << endl;
    }

    A(int v=0):val(v), id(++nextid){
      print("A(int)");
    }

    A(const A& a):val(a.val), id(++nextid){
      print("A(A&)");
    }

    ~A(){
      print("~A()");
    }

    operator int (){
      print("operator int()\n");
      return val;
    }

    A operator() (const A& a){
      print("operator(A&)\n");
      return A(val+a.val);
    }
};

int A::nextid=0;

int main(){
  cout << "main:\n";
  A a(100);
  cout << a+A(a(a)) << endl;
  cout << "end of main\n";
}
```

Q4. Does the following program compile successfully. If yes, show the output or else explain why.

```
#include <iostream>
#include <string>
#include <typeinfo>
using std::string;
using namespace std;

class A {
        int x;
        string name;
        public:
                A(int i=1, string s="A"):x(i),name(s){cout << "A::A(int,string)\n";}
                void print(){cout << "A::x == " << x << " A::s == " << name << endl << endl;}
                virtual A* operator ()(int) =0;
                virtual ~A(){cout << "~A "; print();};
};

class B : virtual public A{
        public:
                B(int i=2, string s = "B"):A(i,s){ cout << "B::B(int,string)\n"; print();}
                virtual A* operator ()(int i){return new B(i, "B::op()");}
                virtual ~B(){cout << "~B "; print();}
};

class C : public B{
        public:
                C(int i=3, string s = "C"):A(i,"From C:"+s),B(-i,"From B:"+s){
                        cout << "B::B(int,string)\n"; print();
                }
                virtual A* operator ()(int i){return new C(i, "C::op()");}
                virtual ~C(){cout << "~C "; print();}
};

C w(42, "Hello C++ World");
B v(5.256);
int main(){
        cout << "Main:\nONE:\n";
        A* a = w(1);
        a->print();
        v(2)->print();
        cout << "TWO:\n";
        B* b = dynamic_cast<B*>(w(2));
        b->print();
        cout << "THREE:\n";
        B b2 = *b;
        b2.print();
        cout << "main ends\n";
}
```

Q5. Consider the following code that is a variation from what presented in class, is it going to compile? If yes, show its output if possible and if not possible describe it, else modify it so it can compile and then show or describe its output. Explain your answer.

```
#include "common.h"
#define EMPLOYEE_COUNT 10

typedef int (*garbage_collector ) (employee_t ** employee_list); void delete_selected_employee
( employee_t ** employee_list, garbage_collector callback) {
        int i = callback(employee_list);
        if ( i != -1 && employee_list[i]!= NULL)
        {
                delete (employee_list[i]);
                employee_list[i] = NULL;
                std::cout << "Deleted employee :"<< i << "\n";
        } else std::cout << "Error Deleting employee :"<< i << "\n";
}

int main () {
        employee_t ** employee_list = (employee_t **)
        calloc(EMPLOYEE_COUNT,sizeof(employee_t *));
        for ( int i = 0 ; i < EMPLOYEE_COUNT ; i++) employee_list[i] = new employee_t();
        delete_selected_employee(employee_list,[employee_list]() -> int {
                for ( int i = 0 ; i < EMPLOYEE_COUNT ; i++ )
                {
                        int index = random_number(0,EMPLOYEE_COUNT);
                        if (index < EMPLOYEE_COUNT && employee_list[index]!= NULL )
                                                                        return index;
                }
                return -1;
        });

        for ( int i = 0 ; i < EMPLOYEE_COUNT ; i++)
                if (employee_list[i] != NULL) delete(employee_list[i]);
        free (employee_list);
        return 0;
}
```

Q6. State the major and minor elements of the Object Model and explain briefly each one of them.

Q7. Explain one difference between inlines and macro functions and support your answer with an example.

Q8. You are asked to develop an array class that allows for dynamic sizing of the array on demand. That is, if the array was originally allocated to contain SIZE elements and an attempt is made to access elements beyond its end, then the array will be expanded automatically by the needed extra size or MINEXPAND whichever is larger (e.g., if an array is allocated SIZE elements and we try to access the SIZE+n element then the additional size will be max(n, MINEXPAND)). In other words you cannot extend the array by less than MINEXPAND elements. Furthermore, we want to avoid the overhead of copying elements every time we expand the array (normally, you would allocate a new space large enough to hold the entire array after expansion, copy the old elements into it, and then delete the old space, so we want to avoid all the copying). Instead, we want to allocate the extra space and somehow join it to the existing one(s) and simply use it (Obviously, you are not allowed to use realloc). You are asked to define the class DArray (you may assume the array will be used to hold ints) as outlined above. Make sure your class implementation provides default and copy constructors, destructor, assignment operator, and index operator (operator[]). Clearly, copying is unavoidable for the copy constructor and assignment operator. You may defined other classes to be used by DArray.

```
//Example of using Darray
const int SIZE=32;
const int MINEXPAND=SIZE/2;

class DArray { … };

int main(){
        DArray a(SIZE); // array of SIZE elements
        for (int i=0; i< 200; i++) //note that: 200>SIZE
                a[i] = i*i;
        for (int i=199; i>=0; --i)
                cout << a[i] << endl;
}
```