Université d'Ottawa
Faculté de génie

École de science
d'informatique
et de génie électrique

University of Ottawa
Faculty of Engineering

School of Electrical
Engineering
and Computer Science

uOttawa

L'Université canadienne
Canada's university

CSI2372A
Advanced Programming Concepts with C++

MIDTERM EXAMINATION

Length of Examination: 75 minutes **November 4, 2015, 14:30**

Professor: Jochen Lang **Page 1 of 14**

Family Name: _____

Other Names: _____

Student Number: _____

Signature _____

You are allowed **ONE TEXTBOOK** as a reference.

No calculators or other electronic devices are allowed.

Please answer the questions in this booklet. If you do not understand a question, clearly state an assumption and proceed.

At the end of the exam, when time is up:
Stop working and turn your exam upside down.
Remain silent.

Question	Marks	Maximum
A.1-A.6		6
2a		2
2b		2
2c		2
2d		2
2e		3
2f		4
3		5
Total		26

PART A: SHORT QUESTIONS (6 MARKS)

1. What is printed by the following program?

```
int a{2};
bool b{false};
if (a&&b) {
  int c = a&b;
  cout << c << endl;
} else {
  int c = a|b;
  cout << c << endl;
}
```

2

2. Given the following definitions

```
class A {
public:
   virtual void print();
};
class B : public A { // omitted };
class C : public A { // omitted };
```

consider the code snippet:

```
A* a;
if ( someVariable > 0 ) {
  a = new B();
} else {
  a = new C();
}
B* b = a; // wrong! Replace this line by downcasting the object *a to *b but you also have to
          // check if the downcast succeeds
```

```
B* b = dynamic_cast<B*>(a);
if ( b==nullptr) cerr << "Failure";
```

3. Consider the following two functions:

```
void makePair( int a, int b, int* p) {
            p[0] = a;
            p[1] = b;
}
void makePair( double a, double b, double* p) {
            p[0] = a;
            p[1] = b;
}
```

Define a function template replacing the functions above which can be instantiated correspondingly. For example:

```
int a, b, p[2];
makePairTemp( a, b, p );
double x, y, z[2];
makePairTemp( x, y, z );
```

```
template<typename T>
void makePairTemp( T a, T b, T* p ) {
  p[0] = a;
  p[1] = b;
}
```

---

4. Create a local array (on the stack) of two pointers to integer and then for each of the pointers allocate dynamically an array of 10 integers (on the heap).

```
int* array[2];
array[0] = new int[10];
array[1] = new int[10];
```

---

5. Given the function declarations below

```
void func(const float a, const float& b, float *c ) {
  cout << "void func( " << a << ", " << b << ", " << *c  << " )" << endl;
}
```

Call the above function such that it prints: `void func(1, 2, 3);`

```
float a{1.},b{2.},c{3.};
```

---

func(a,b,&c);

---

6. There is one compile error in the program below, identify it.

```
#include <iostream>

struct A {
  void func() {
    std::cout << "A.func()" << std::endl;
  }
};

class B : protected A {
   void func() { A::func();}
};

class C : private B {
public:
   void func() { A::func();}
};

int main() {
  A a; B b; C c;
  a.func();
  b.func();
  c.func();
  return 0;
}
```

[Error] 'void B::func()' is private

PART B: PROGRAMMING QUESTIONS (20 MARKS)

2. Consider the following definitions of the class Time using a 24 hr clock.

```
class Time {
  unsigned char d_hour = 0;
  unsigned char d_minutes = 0;
public:
  inline Time();
  inline Time(unsigned char _hour, unsigned char _minutes);
  inline void get(unsigned char& _hour,
                  unsigned char& _minutes) const;
  inline void set(unsigned char _hour, unsigned char _minutes);
  inline void print() const {
   cout << static_cast<int>(d_hour) << ":" <<
           static_cast<int>(d_minutes);
  }
};
```

Leading to the following WorkWeek definition which uses growable, dynamically allocated arrays of start and end times of work shifts.

```
class WorkWeek {
  Time *d_start;
  Time *d_end;
  int d_size;
  int d_currNum = 0; // next available index to add a shift
public:
  WorkWeek(int _nShifts); // Part 2.a
  WorkWeek(const WorkWeek& _w); // Part 2.b
  ~WorkWeek(); // Part 2.c
  void print() const; // Part 2.d
  Time getTotalHours() const; // Part 2.e
  void addShift(unsigned char _hour,
                unsigned char _minutes, unsigned int _duration);
  // Part 2.f
};
```

a. Define a constructor that dynamically allocates the Time arrays d_start and d_end. [2]

```
WorkWeek::WorkWeek(int _nShifts) : d_size(_nShifts)
{
  assert( _nShifts > 0 );
  d_start = new Time[d_size];
  d_end = new Time[d_size];
}
```

b. Define a copy constructor implementing a deep copy strategy [2]:

```
WorkWeek::WorkWeek(const WorkWeek& _w ) : d_size(_w.d_size),
d_currNum(_w.d_currNum) {
    d_start = new Time[d_size];
    d_end = new Time[d_size];
    for (int i=0; i<d_currNum; ++i ) {
            d_start[i] = _w.d_start[i];
            d_end[i] = _w.d_end[i];
    }
}
```

c. Define the destructor for WorkWeek [2].

```
WorkWeek::~WorkWeek() {
    if (d_size>0) {
            delete[] d_start;
            delete[] d_end;
    }
}
```

d. Define the print function for WorkWeek which should print all shifts currently stored in WorkWeek by calling the print function for Time [2]. For example, a shift starting at 8:30 and ending at 12:50 should print in a single line as: `8:30 to 12:50`

```
void WorkWeek::print() const {
    for (int i=0; i<d_currNum; ++i ) {
            d_start[i].print();
            cout << " to ";
            d_end[i].print();
            cout << endl;
    }
}
```

e. Define the function getTotalHours that counts up the total time for all shifts stored in WorkWeek [3].

```
Time WorkWeek::getTotalHours() const {
     // do the calculation in minutes
     int minutes{0};
     for (int i=0; i<d_currNum; ++i ) {
          unsigned char startHrs, startMin, endHrs, endMin;
          d_start[i].get( startHrs, startMin );
          d_end[i].get( endHrs, endMin );
          // rely on integral promotion
          minutes += endHrs*60 + endMin - startHrs*60 -
                    startMin;
     }
     // Convert minutes in hours + minutes
     return Time( minutes/60, minutes%60 );
}
```

f. Define the function addShift for WorkWeek which adds a new shift at the next available slot. If the arrays in WorkWeek are too small, create new arrays twice the size of the old array (growable array strategy) [4].

```
void WorkWeek::addShift(unsigned char _hour, unsigned
char _minutes, unsigned int _duration ) {
     if (++d_currNum == d_size ) {
          assert(d_size > 0);
          // grow 2x
          d_size *= 2;
          Time *start = new Time[d_size];
          Time *end = new Time[d_size];
          // copy
          for (int i=0; i<d_currNum; ++i ) {
               start[i] = d_start[i];
               end[i] = d_end[i];
          }
          // swap
          delete[] d_start;
          delete[] d_end;
          d_start = start;
          d_end = end;
     }
     d_start[d_currNum-1] = Time( _hour, _minutes );
     int minutes = (_minutes + _duration)%60;
     int hour = _hour + (_minutes + _duration)/60;
     d_end[d_currNum-1] = Time( hour, minutes );
     return;
}
```

3. Consider the following program and specify what is printed to the console [5]

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

class Food
{
  protected:
    double calories;
    std::string name;
  public:
    Food(){ cout << "Food constructor" << endl;}
    ~Food() { cout << "Food destructor" << endl; }
    std::string getName() const { cout << "Food name" << endl;
   return name;}
    virtual double getCalories() const { cout << "Food
   calories" << endl; return calories;}
};

class Cake : public Food
{
  private:
    double weight;

  public:
    Cake() { cout << "Cake constructor" << endl; }
   ~Cake(){ cout << "Cake destructor" << endl; }
    std::string getName() const {
      cout << "Cake name" << endl; return name;
    }
    double getCalories() const {
      cout << "Cake calories" << endl; return calories;
    }
};

void fct(const Food &d) {
    Cake c;
    return;
}

int main()
{
    Food *food;
    Cake *cake;

    std::cout << "[1]" << std::endl;
    food= new Cake();

    std::cout << "[2]" << std::endl;
    Cake cakes[3];
```

```
    std::cout << "[3]" << std::endl;
    cake= dynamic_cast<Cake*>(food);

    std::cout << "[4]" << std::endl;
    food->getCalories();
    food->getName();
    std::cout << "[5]" << std::endl;
    cake->getCalories();
    cake->getName();

    std::cout << "[6]" << std::endl;
    cakes[0]= *cake;

    std::cout << "[7]" << std::endl;
    fct(cake[1]);

    std::cout << "[8]" << std::endl;
    delete food ;
    std::cout << "[9]" << std::endl;
}
```

```
[1]
Food constructor
Cake constructor
[2]
Food constructor
Cake constructor
Food constructor
Cake constructor
Food constructor
Cake constructor
[3]
[4]
Cake calories
Food name
[5]
Cake calories
Cake name
[6]
[7]
Food constructor
Cake constructor
Cake destructor
Food destructor
[8]
Food destructor
[9]
Cake destructor
Food destructor
Cake destructor
Food destructor
Cake destructor
Food destructor
```