Université d'Ottawa
Faculté de génie

School of Electrical Engineering and Computer Science

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

École de science informatique et de génie électrique

CSI2372 Advanced Programming Concepts with C++

FINAL EXAM

Length of Examination: 3 hours **December 19, 2016, 9:30**

Professor: Jochen Lang **Page 1 of 22**

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 close your exam booklet. Remain silent.

Question	Marks	Out of
A.1		2
A.2		2
A.3		1
A.4		1.5
B		8
C.1		3
C.2		2
C.3		4
C.4		1
C.5		1
C.6		2
D		10.5
Total		38

Part A: Short Questions (6.5 marks)

1. What is printed by the following program? [2]

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

class A {
protected:
   int d_a = 1;
public:
  virtual A& operator+=(int i) {
         d_a += i;
         cout << "A: " << d_a << endl;
         return *this;
   }
};

class B : public A {
public:
  virtual B& operator+=(double d) {
         d_a += 2*static_cast<int>(d);
         cout << "B: " << d_a << endl;
         return *this;
   }
};

int main () {
   B ab;
   A& a = ab;
   A aa = ab;
   A* aptr = &ab;
   ab += 1;
   a += 1.0;
   (*aptr) += 1;
   aa += 1;
   return 0;
}
```

2. What is printed by the following program? [2]

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

struct Error {
  int num;
};

struct CodeError: public Error {
  int code;
};

struct A {
  A(int n) {
   if (n==1) {
         CodeError erd; erd.num = 999;
         erd.code = 12; throw erd;
   }
  }
};

int main() {
  try {
    A a(1); cout << "Created a(1)" << endl;
  }
  catch (Error e) {
    cout << "exception Error: " << e.num << endl;
  }
  catch (CodeError e) {
    cout << "exception CodeError: " << e.num << " " << e.code << endl;
  }
  catch (...) {
    cout << "Unknown Error"  << endl;
  }
  cout << "after" << endl;
  try {
    A b(1); cout << "Created b(1)" << endl;
  }
  catch (CodeError e) {
    cout << "exception CodeError: " << e.num << " " << e.code << endl;
  }
  catch (Error e) {
    cout << "exception Error: " << e.num << endl;
  }
  return 0;
}
```

(Please answer on next page).

3. What is printed by the following program? [1]

```
#include <iostream>
#include <list>

using namespace std;

using LISTINT=list<int>;

int main() {
  LISTINT li;
  li.push_front (2);
  li.push_front (1);
  li.push_back (3);
  for( auto i=li.cbegin(); i!=li.cend(); ++i)
    cout<<*i<<" ";
  cout<<endl;
  for(auto i=li.rbegin(); i!=li.rend(); ++i)
    cout<<*i<<" ";
  cout<<endl;
  return 0;
}
```

4. What is printed by the following program [1.5]?

```
#include <iostream>
#include <deque>
#include <algorithm>

using namespace std ;

template <class T>
void print (T& container) {
   for ( auto& e : container ) cout << e << " ";
   cout << endl;
   return;
}

int main() {
  char mot[] = {"xyz"};
  deque<char> pl(mot, mot+3); print(pl);
  pl.push_front('a'); print(pl);
  pl[2] = '+';
  pl.push_front('b');
  pl.pop_back(); print(pl);
  auto ip = find (pl.begin(), pl.end(), 'x');
  pl.erase(pl.begin(), ip); print(pl);

  return 0;
}
```

Part B: Abstract Data Types (8 MARKS)

The following listing contains global and in class operators for the given class Hike. A hike has a distance and an elevation (gain).

A `main` function and the expected print out is given at the end of the declarations to help clarify the intended functionality of the operators and the class.

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

class Hike;
// Global operators declarations
ostream& operator<<( ostream&, const Hike& );
istream& operator>>( istream&, const Hike& );
Hike operator+( const Hike&, const Hike& );
Hike operator+( const Hike&, float);
Hike operator+( const Hike&, int);

class Hike {
   float d_distance; // distance in kilometers
   int d_elevation; // elevation gain in meters
public:
   Hike() = default;
   explicit Hike(float distance, int elevation=0);

   Hike& operator+=( Hike& );
   Hike& operator++();
   Hike operator++(int);

   inline operator float() const;
   inline operator int() const;

   void imperial( ostream& os ) const;
   friend ostream& operator<<( ostream&, const Hike& );
   friend istream& operator>>( istream&, Hike& );

protected:
   // convert distance from kilometers to miles; 1 mile = 1.609 km
   static float convert( float );
   // convert elevation gain from meters to feet; 1 foot = 0.3m
   static int convert( int );
};
```

```
int main() {
   Hike walk(12.5f,350), up(25.0f,780), down(6.3f,-300);

   cout << walk << endl;
   up += down;              cout << up << endl;
   Hike upLong = up + 3.3f; cout << upLong << endl;
   Hike upup = up + 750;    cout << upup << endl;

   cout << "Enter a walk: ";   cin >> walk;
   cout << walk << endl;

   walk.imperial( cout );
   return 0;
}

/* Console Output
12.5 350
31.3 480
34.6 480
31.3 1230
Enter a walk: 23.5 120
23.5 120
Hike: 14.6053 400 */
```

Complete the code starting on the next page in the space provided. (Note if you need less lines that is perfect, needing more likely indicates a problem).

```
// initialize distance and elevationGain
  Hike::Hike(float distance, int elevation) :
     ____________________________________________

     ____________________________________________
  {}


  // add distances and elevationGain together
  Hike& Hike::operator+=( Hike& h) {

     ____________________________________________

     ____________________________________________

     return _____________________________________________;
  }


  // increase the elevation gain by 1
  Hike& Hike::operator++() {

     ____________________________________________

     return _____________________________________________;
  }

  // increase the elevation gain by 1
  Hike Hike::operator++(int) {

     ____________________________________________

     ____________________________________________

     return _____________________________________________;
  }


  // return hike distance as float
  Hike::operator float() const {

     return _____________________________________________;
  }
```

```
 // return hike elevation as int
Hike::operator int() const {

   return ____________________________________________________;
}


// stream insertion and extraction
// read/print distance and elevation gain directly
void Hike::imperial( ostream& os ) const {
   os << "Hike: " << convert(d_distance) << " "
   << convert(d_elevation) << endl;
}

// static protected methods
// convert distance from kilometers to miles; 1 mile = 1.609 km

_____________________________________________ convert( float km){

   return km / 1.609;
}

// convert elevation gain from meters to feet; 1 foot = 0.3m

_____________________________________________convert( int e) {

   ____________________________________________________

   ____________________________________________________

   return ____________________________________________________;
}


// stream insertion operator
ostream& operator<<( ostream& os, const Hike& h) {

   ____________________________________________________


   return ____________________________________________________;
}
```

```
// stream extraction operator
istream& operator>>( istream& is, Hike& h) {

   _____________________________________________

   if ( !is ) _____________________________________________

   return _____________________________________________;

}

// add the distance and elevation gain of two hikes
Hike operator+( const Hike& a, const Hike& b) {

   _____________________________________________

   _____________________________________________

   return    _____________________________________________;

}

// add distance to a hike
Hike operator+( const Hike& h, float d) {

   _____________________________________________

   _____________________________________________

   return    _____________________________________________;

}

// add elevation to a hike
Hike operator+( const Hike& h, int e) {

   _____________________________________________

   _____________________________________________

   return    _____________________________________________;

}
```

PART C: Constructors and Assignments (13 MARKS)

Consider the following parent and derived class which implement internal aggregation.

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

class PlayList {
protected:
      struct SongEntry {
            string d_artist;
            string d_title;
            SongEntry* d_next;
      };
      int d_size = 0;
      SongEntry* d_list = nullptr;
public:
      PlayList() = default;
      PlayList( const PlayList& );
      virtual ~PlayList();
      PlayList& operator=( const PlayList& );
};

class Top5 : public PlayList {
public:
      Top5() = default;
      Top5( const Top5& );
      ~Top5();
      Top5& operator=( const Top5& );

      bool append( string artist, string title) {
            if  ( d_size >= 5 ) return false;
            SongEntry* d_tmp = d_list;
            d_list = new SongEntry{ artist, title, d_tmp };
            ++d_size;
            return true;
      }

      bool remove() {
            if (d_size<=0) return false;
            SongEntry* d_tmp = d_list->d_next;
            delete d_list;
            d_list = d_tmp;
            --d_size;
            return true;
      }

};
```

1. Implement the copy constructor for the class `PlayList`. [3]

```
PlayList( const PlayList& )
```

2. Implement the destructor for the class `PlayList` [2]?

```
virtual ~PlayList() {
```

3. Implement the assignment operator for the class `PlayList` [4]

```
PlayList& operator=( const PlayList& );
```

4. Implement the copy constructor for the class `Top5`. [1]

```
Top5( const Top5& )
```

5. Implement the destructor for the class `Top5` [1]?

```
~Top5() {
```

6. Implement the assignment operator for the class `Top5` [2]

```
Top5& operator=( const Top5& );
```

Part D: Containers of the Standard Template Library (10.5 marks)

The following class template adapts a `std::vector` to implement a singly-linked list. You will need to use the standard template library algorithm `std::find` for your implementations of the functions `index_of` and `iter`.

Complete the code below:

```
#include <iostream>
#include <vector>
#include <list>
#include <algorithm>

using namespace std;

template <class T>
class SLL : protected std::vector<T> {

public:
      // add at the end of the vector
      void add(T& entry);
      // return the element at the end of the vector
      T peek();
      // return and remove element at the end of the vector
      T get();
      // clear the whole vector
      void clear();
      // find the index in the vector of an element or return -1
      // assumes T has an equality operator
      size_t index_of( const T& );
      // find an element and return an iterator to it; returns
      // an iteraor to end on failure
      // assumes T has an equality operator
      typename std::vector<T>::const_iterator iter( const T& );

      // return this SLL as a std::list
      std::list<T> toList();

};
```

std::find, std::find_if, std::find_if_not

Defined in header `<algorithm>`

```
template< class InputIt, class T >
InputIt find( InputIt first, InputIt last, const T& value );                                   (1)

template< class ExecutionPolicy, class InputIt, class T >
InputIt find( ExecutionPolicy&& policy, InputIt first, InputIt last, const T& value );   (2) (since C++17)
```

Returns the first element in the range `[first, last)` that satisfies specific criteria:

1) `find` searches for an element equal to `value`
2) Same as (1), but executed according to `policy`. These overloads do not participate in overload resolution unless std::is_execution_policy_v<std::decay_t<ExecutionPolicy>> is true

Parameters

first, last	-	the range of elements to examine
value	-	value to compare the elements to
policy	-	… *Omitted* …

Type requirements

- `InputIt` must meet the requirements of `InputIterator`.
- `UnaryPredicate` must meet the requirements of `Predicate`.

Return value

Iterator to the first element satisfying the condition or `last` if no such element is found.

1. Implement the add function that adds an entry at the end of SLL's vector. [1]

```
template <class T>
void SLL<T>::add(T& element) {
    _____________________________________________________

    _____________________________________________________
}
```

2. Implement the peek function that returns the last added element (but does not remove it).[1]

```
template <class T>
T SLL<T>::peek() {

    _____________________________________________________

    _____________________________________________________
}
```

3. Implement the get function that removes and returns the last added element [1.5]

```
template <class T>
T SLL<T>::get() {
    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

}
```

4. Implement the clear function that clears all elements from this SLL's vector [1].

```
template <class T>
void SLL<T>::clear() {

    _____________________________________________________

    _____________________________________________________
}
```

5. Implement the function `index_of` that finds the first entry equal to the argument and returns the index of it in the vector. If the entry is not found return -1. Your implementation should assume that T has an equality operator. You will need to use the standard template library algorithm `std::find` [3].

```
template <class T>
size_t  SLL<T>::index_of( const T& e ) {

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________


    return _____________________________________________________;
}
```

6. Implement the function `iter` that finds the first entry equal to the argument and returns an iterator to it. If the entry is not found return an iterator to the end. Your implementation should assume that T has an equality operator. You will need to use the standard template library algorithm `std::find` [2].

```
template <class T>
typename std::vector<T>::const_iterator SLL<T>::iter( const T& e ) {
    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________


    return _____________________________________________________;
}
```

7. Implement the `toList` function that makes a copy of the SLL's vector as a `std::list` [1].

```
template <class T>
std::list<T> SLL<T>::toList() {

    _____________________________________________

    _____________________________________________


   return _____________________________________________;
}
```