CMPUT 201 Final Exam Instructor: Michael Buro

December 13, 2002 9:00-11:00am

1: / 8	***3 : /6***	***5: /12***	***7: /6***	***9: /12***	***total /86***
2: /6	**4: /6**	**6: / 20**	**8: /16**	**10: /6**	

Instructions:

- This exam is closed book. No conversations, please. Cheating is lame and may have unpleasant consequences.
- **Print** your name and student id on **all** page headings.
- Put your **OneCard** on your desk – it will be checked.
- Write your answers **legibly** in the space below or next to the questions. Use a pen.
- You can use the back sides as scratch space - no other sheets are accepted.
- Skip questions you can't answer immediately and return to them later.
- The total number of marks is 86

1. Write a function that rotates an unsigned int x k bits to the right (No STL!) (8 marks)

```
void rotate_right(unsigned int& x, int k)
{
  assert(k >= 0);









}
```

2. Compute the value of the following C expressions. (6 marks)
   Think before you plunge into long calculations! (int x = 101)

   A) (x << 31) >> 31
   B) (x > (x | 77)) + (x <= (x | 77))
   C) (~x+1) + x
   D) x >> (x-1)
   E) x % 1
   F) (x & 0x1F) | 31

3. How many bytes in memory do the following variables occupy on machines where each byte in memory has a 32-bit address? (6 marks)

   A) char ***x;
   B) bool *x[20];
   C) unsigned short (*x)(char *);
   D) struct X { signed short a[24]; bool b[8]; } x;
   E) class Y { public: int c; float f(); virtual void g(); virtual ~Y(); } x;
   F) union { char a; float b; double *c; char (*d)(double *); } x;

4. Write a const member function that checks in time linear in the number of list elements whether a singly-linked list has a cycle. The function is not allowed to allocate heap memory. The end of a list is indicated by succ == 0.
   (6 marks)

```
template <class T> class List {
public:
  T data;
  List *succ;
  bool has_cycle() const;
  ~List();
};

template <class T> bool List<T>::has_cycle() const
{




}
```

5. A) Complete the implementation of the STL **for_each** template function below.
for_each
applies a unary functor to all elements in range [first, last) . The results of the functor are ignored and the functor is returned. (4 marks)

```
template <class InputIterator, class UnaryFunction>
UnaryFunction for_each(InputIterator first,InputIterator last,
                       UnaryFunction f)
{









}
```

B) Implement the STL **set_intersection** template function that outputs common elements of two given sorted ranges [first1, last1) and [first2, last2) to an output iterator which is also returned. Use operators <,>,==... to compare elements. (8 marks)

```
template<class InputIter1, class InputIter2, class OutputIter>
OutputIter set_intersection(InputIter1 first1, InputIter1 last1,
                            InputIter2 first2, InputIter2 last2,
                            OutputIter result)
{


















}
```

6. The following definition of template class Tree - which implements a binary tree - is incomplete. Implement the missing constructor, assignment operator, size(), free(), and copy() functions. [Tips: “Think recursively”, use copy() in assign.op.] (total: 20 marks)

```
template <class T> class Tree {
public:
  T data;
  Tree *left, *right; // pointers to left and right successor

  Tree();                                 // creates tree with one node
  ~Tree() { free(); }                     // destroys entire tree
  Tree(const Tree& x) { copy(x); }       // copy constr.: deep copy!
  T& operator=(const Tree& x);  // assignment operator: deep copy!

  int size() const;          // number of nodes in tree

protected:
  void free();               // destroys both subtrees
  void copy(const Tree& x); // *this = deep copy of x
                             // precond.: *this has no successors
};
```

```
template<class T> Tree::Tree() {                                (2 marks)






}
```

```
template<class T> T& Tree<T>::operator=(const Tree& x)     (4 marks)
{














}
```

```
template<class T> int Tree<T>::size() const
{

}
```

(4 marks)

```
template<class T> void Tree<T>::free()
{

}
```

(4 marks)

```
template<class T> T& Tree<T>::copy(const Tree& x)
{

}
```

(6 marks)

7. What are the g++ command line options for the following tasks: (6 marks)

A) Generate execuable foo from foo.c that uses functions from the math-lib.
B) Compile test.c with optimization level 3 and create object file test.o
C) Create a.out from test.c for profiling purposes
D) Create a.out from test.c for debugging purposes
E) Generate test.s from test.c to study the assembly language output.
F) Link test1.o and test.o and generate exectuable test.

8. Given are the following class definitions and pointer variables:
(16 marks)

```
class X {
  void g(X x);
  void h(X &x);
protected:
  int a;
public:
  X() { a = 0; }
  virtual void u() { cout << ”X”; }
};
```

```
class Y : public X {
public:
  Y() { p = new int[100]; }
  void u() { cout << “Y”; }
  ~Y() { delete [] p; }
};

Y *py = new Y; X *px = py;

int main() { ... }
```

Determine whether the following statements are true (T) or false (F). [READ! One mark for each correct answer; <u>three</u> wrong answers are <u>free</u>; one mark is deducted for additional wrong answers; <u>not answering</u> is an option resulting in 0 marks for that question; mark total >= 0]

A) X::g is visible in Y T F

B) X::u can call X::h T F

C) X::a is visible in Y T F

D) Y::u can call X::u T F

E) px->a = 0 allowed in main T F

F) X::a = 0 allowed in Y::u T F

G) py->a = 0 allowed in main T F

H) sizeof(X) == sizeof(Y)+4 T F

I) delete px; in main works correctly T F

J) delete py; in main works correctly T F

K) px->u(); in main outputs “X” T F

L) Storing auto_ptrs in containers is a “no-no” T F

M) Base-class constructors are called at the end of derived class constructors T F

N) Exceptions can be ignored T F

O) Throwing exceptions in catch-blocks is forbidden T F

P) Default constructors initialize data members with 0 T F

9. What is the worst case run-time complexity (measured in the number N of elements in the containers or ranges) for the following STL operations. Options are **C** (constant), **Log(N)** (logarithmic), **N** (linear), **N*Log(N)** (log-linear) (total: 12 marks)

A) vector::push_back(x)
B) vector::insert(pos, x)
C) list::push_front(x)
D) list::find(x)
E) set::insert(x)
F) set::find(x)

G) map::insert(key, x)
H) map::erase(key)
I) sort(first, last);
J) set_union(first1, last1, first2, last2, result)
K) find(first, last, x)
L) binary_search(first, last, x)

10. Implement the missing random access write_at_loc function of class Data below and squish the bug present in the write function. (Hints: is sizeof() really reporting the total size of all data members? What about padding? The prototype of fwrite is
int fwrite(void *ptr, int size, int nelem, FILE *stream); its return value is the numbef of elements written) (total: 6 marks)

```
class Data {
public:
  short x;
  char a[20];

  Data();
  virtual ~Data();

  // write *this in binary format at the current file position
  // return true iff something went wrong                          (4 marks)

  bool write(FILE *out) {

    return fwrite(this, sizeof(*this), 1, out) == 0;

  }

  // write *this in binary format at record(!) location recloc
  // return true iff something went wrong                          (2 marks)

  bool write_at_loc(FILE *out, int recloc) {









  }
};
```