IS0020 Program Design and Software Tools
Midterm, Feb 24, 2004

Name:

Instruction

There are two parts in this test. The first part contains 50 questions worth 80 points. The second part constitutes 20 points and there are four questions, but you have to attempt only two. There are also some bonus points you can get. Maximum time allowed is 2 hours and 30 minutes.

Good Luck!

Part I:
There are 50 question and the total marks is 80

1. Which of the following statements about C++ is false?
   (a) C++ is an extension of C.
   (b) C++ provides capabilities of *object-oriented programming*.
   [✓ (c) C++ is an interpreted language.]
   (d) C++ is a compiled language.

2. Using standard library functions can be more efficient because ___________.
   (a) they save programming time
   (b) they are carefully written to perform optimally
   (c) they increase program portability
   [✓ (d) all of the above.]

3. The *compile* stage is when _____________.
   (a) the object code is linked with code for functions in other files
   [✓ (b) the C++ program is translated into machine language code]
   (c) the program is executed one instruction at a time
   (d) the program is placed in memory

4. Which operation returns the integer remainder when 15 is divided by 6?
   (a) `15 / 6`
   [✓ (b) `15 % 6`]
   (c) `15 ^ 6`
   (d) `15 * 6`

5. Which of the following encompasses the other three?
   (a) sequence structure
   (b) repetition structure
   [✓ (c) control structure]
   (d) selection structure

6. Which of the following will not increment variable `c` by one?
   [✓ (a) `c + 1;`]
   (b) `c++;`

(c) `++c;`
(d) `c += 1;`

7. In which of the following is **`y`** not equal to **`5`** after execution? Assume **`x`** is equal to **`4`**.
(a) `y = 5;`
(b) `y = x++;`
(c) `y = ++x;`
(d) `y = x = 5;`

8. The statement
```
      while ( --counter >= 1 )
            counter % 2 ? cout << “A” : cout << “B”;
```
cannot be rewritten as
(a)
```
while ( --counter >= 1 )
      if ( counter % 2 )
            cout << “A”;
      else
            cout << “B”;
```
(b)
```
while ( counter >= 1 )
      if (counter % 2)
            cout << “A”;
      else
            cout << “B”;
      --counter;
```
(c)
```
while ( counter > 1 )
      {
            --counter;
            if ( counter % 2 )
                  cout << “A”;
            else
                  cout << “B”;
      }
```
(d)
```
do
      {
            --counter;
            cout << ( counter % 2 ? “A” : “B” );
  } while ( counter >= 2 );
```

9. Consider the loop
```
for ( int x = 1; x < 5; increment )
      cout << x + 1 << endl;
```
If the last value printed is **`5`**, which of the following might have been used for ***`increment`***?
(a) `x++`
(b) `x += 1`
(c) `++x`
(d) Any of the above

10. If a **`do/while`** structure is used,
(a) an infinite loop will not take place
(b) counter-controlled repetition is not possible
(c) the body of the loop will execute at least once
(d) an off-by-one error will not occur

11. Which of the following will not increment variable **`c`** by one?
(a) `c + 1;`

(b) **c++;**
(c) **++c;**
(d) **c += 1;**

12. The following program segment will

```
int counter = 1;
do {
      cout << counter << " ";
} while ( ++counter <= 10 );
```

(a) print the numbers 1 through 11
(b) print the numbers 1 through 10
(c) print the numbers 1 through 9
(d) cause a syntax error

13. The expression

```
      if ( num != 65 )
```

cannot be replaced by:
(a) **if (num > 65 || num < 65)**
(b) **if (!(num == 65))**
(c) **if (num – 65)**
(d) **if (!(num – 65))**

14. Variables are also known as
(a) *lvalues*, but can be used as *rvalues*
(b) *lvalues*, and cannot be used as *rvalues*
(c) *rvalues*, and cannot be used as *lvalues*
(d) constant variables

15. Consider the following code, assuming that **x** is an integer variable with an initial value of **12**:

```
      if( x = 6 )
            cout << x;
```

What is the output?
(a) 6
(b) 12
(c) nothing
(d) a syntax error is produced

16. An identifier's storage class
(a) determines the period during which that identifier exists in memory
(b) determines whether an identifier is known only in the current source file or in any source file with proper declarations
(c) determines where the identifier can be referenced in a program
(d) all of the above

17. Which of the following is not true of **static** local variables?
(a) they are accessible outside of the function in which they are defined
(b) they retain their values when the function is exited
(c) they are initialized to zero if not explicitly initialized by the programmer
(d) they can be of type **int**

18. What value does function **whatDoIOutput** return when called with a value of 4?

```
int whatDoIOutput ( int number ) {
      if ( number <= 1 )
            return 1;
      else
            return number * whatDoIOutput( number – 1 );
}
```

(a) 1
(b) 24
(c) 0
(d) 4

19. A reference parameter
(a) is an alias for its corresponding argument
(b) is declared by following the parameter's type in the function prototype by an ampersand (**&**)
(c) cannot be modified
(d) both (a) and (b)

20. **Call-by-reference can achieve the security of call-by-value when**
(a) the value being passed is small
(b) a large argument is passed in order to improve performance
(c) a pointer to the argument is used
(d) the **`const`** qualifier is used

21. Given the following function template
```
template < class T >
T whatAmI( T value1, T value2 )
{
   if ( value1 > value2 )
      return value2;
   else
      return value1;
}
```
what would be returned by the following two function calls?
```
whatAmI( 2, 5 );
whatAmI( 2.3, 5.2 );
```
(a) 5, a type-mismatch error
(b) 5, 5.2
(c) 2, 2.3
(d) two error messages

22. To prevent modification of array values in a function
(a) the array must be declared **`static`** in the function
(b) the array parameter can be preceded by the **`const`** qualifier
(c) a copy of the array must be made inside the function
(d) the array must be passed call-by-reference

23. Assuming that `int a` has a value of `3` and that integer array `b` has `7` elements, what is the correct way to assign the value of the sum of `3` and the fourth element, to the fifth element of the array?
(a) `b[ a + 1 ] = b[ a ] + 3;`
(b) `b[ a + 1 ] = b[ a - 1 ] + 3;`
(c) `b[ a ] + 1 = b[ a ] + 3;`
(d) `b[ a + 2 ] = b[ a ] + 3;`

(corrected answer)

24. Three of the following expressions have the same value. Which of the following's value is different from the others?
(a) **`*&Ptr`**
(b) **`&*Ptr`**
(c) **`*Ptr`**

(d) `Ptr`

25. **When a compiler encounters a function parameter for a single-subscripted array of** the form `int a[]`, it converts the parameter to
(a) `int a`
(b) `int &a`
(c) `int * a`
(d) `int * const a`

26. Assuming that `t` is an array and `tPtr` is a pointer to that array, what expression refers to the address of the fourth element?
(a) `*( tPtr + 3 )`
(b) `tPtr[ 3 ]`
(c) `&tPtr[ 3 ]`
(d) `*( t + 3 )`

(Should have been `&tPtr` in (c) and (d))

27. Object-oriented programming generally does not focus on
(a) separating the interface and implementation of a program.
(b) ease of program modifiability.
(c) information hiding.
(d) client-side access to implementation details.

28. Non-`static` member variables declared `private`
(a) can never be accessed directly or indirectly by the client.
(b) can never be modified directly or indirectly by the client.
(c) can be accessed and/or modified by any object of a different class.
(d) can be accessed and/or modified by `public` member functions and by `friend`s of the class.

29. A class may contain multiple constructors if
(a) they have different names.
(b) they have different argument lists.
(c) they have the same argument list.
(d) they have different return types.

30. Given the class definition

```
class CreateDestroy {
      public:
        CreateDestroy() { cout << "constructor called, "; }
        ~CreateDestroy() { cout << "destructor called, "; }
};
```

What will the following program output?

```
int main() {
      for (int i = 1; i <= 2; i++)
            CreateDestroy c2;
      return 0;
}
```

(a) `constructor called, destructor called, constructor called, destructor called,`
(b) nothing
(c) `constructor called, constructor called, destructor called, destructor called,`
(d) `constructor called, constructor called,`

(This question may have been confusing)

31. Returning references to non-**`const private`** data
(a) allows **`private`** functions to be modified.
(b) is only dangerous if the binary scope resolution operator (**`::`**) is used in the function prototype
(c) allows **`private`** member variables to be modified, thus "breaking encapsulation."
(d) results in a compiler error.

32. The code fragment
```
Increment::Increment(int c, int i) : increment (i) {
      count = c;
}
```
tells you
(a) **`Increment`** is a **`const`** variable.
(b) **`Increment`** must be a **`const`** function.
(c) **`increment`** may or may not be a destructor.
(d) **`increment`** may be a **`const`** variable.

33. If the line **`friend class A;`** appears in **`class B`**, and **`friend class B;`** appears in **`class C`** then
(a) **`class A`** is a friend of **`class C`**.
(b) **`class C`** can call **`class A`**'s **`private`** member functions.
(c) **`class A`** can access **`private`** variables of **`class B`**.
(d) **`class B`** can access **`class A`**'s **`private`** variables.

34. If the functions **`a()`**, **`b()`** and **`c()`** all return references to an object **`Test`** (using the **`this`** pointer) and function **`d()`** is declared **`void`**, which of the following statements has correct syntax?
(a) **`a().b().Test;`**
(b) **`Test.d().c();`**
(c) **`Test.a().Test.d();`**
(d) **`Test.a().b().d();`**

35. **`static`** member functions:
(a) can use the **`this`** pointer.
(b) can only access other **`static`** member functions and **`static`** variables.
(c) cannot be called until their class is instantiated.
(d) can be declared **`const`** as well.

36. Proxy classes are best described as an example of
(a) object-oriented programming (as used in the text).
(b) structured programming.
(c) information hiding.
(d) utility functions.

37. Which statement about operator overloading is false?
(a) New operators can never be created.
(b) The precedence of an operator cannot be changed by overloading.
(c) Overloading cannot change how an operator works on built-in types.
(d) Certain overloaded operators can change the number of arguments they take.

38. An overloaded **`+`** operator takes a class object and a **`double`** as operands. For it to be commutative (i.e., **`a + b`** and **`b + a`** both work),
(a) **`operator+`** must be a member function of the class from which the objects are instantiated.
(b) **`operator+`** must be a non-member function.
(c) the **`operator+`** function that takes the object as the left operand must be a member function, and the other **`operator+`** must be a non-member function.
(d) both **`operator+`** functions must be non-member **`friend`** functions of the class.

39. Suppose you have a programmer-defined data type `Data` and want to overload the `<<` operator to output your data type to the screen in the form "`cout << dataToPrint";` and allow cascaded function calls. The first line of the function definition would be
(a) `ostream &operator<<(ostream &output, const Data &dataToPrint)`
(b) `ostream operator<<(ostream &output, const Data &dataToPrint)`
(c) `ostream &operator<<(const Data &dataToPrint, ostream &output)`
(d) `ostream operator<<(const Data &dataToPrint,ostream &output)`

40. `y` and `z` are user-defined objects and the `+=` operator is an overloaded member function. The operator is overloaded such that `y += z` adds `z` and `y`, then stores the result in `y`. Which of the following expressions is equivalent to `y += z`?
(a) `y = (y.operator+=) + (z.operator+=)`
(b) `y.operator+=( z )`
(c) `y = y + z`
(d) `y.operator+=( z ) = y.operator+=( z ) + z.operator+=( z )`

41. There exists a data type `Date` and member function `Increment`. The `++` operator is being overloaded to postincrement an object of type `Date`. Select the correct implementation.
(a)
```
Date Date::operator++( int ) {
      Date temp = *this;
      Increment();
      return *temp;
}
```
(b)
```
Date Date::operator++( int ) {
      Increment();
      Date temp = *this;
      return temp;
}
```
(c)
```
Date Date::operator++( int ) {
      Date temp = *this;
      return this;
      temp.Increment();
}
```
(d)
```
Date Date::operator++( int ) {
      Date temp = *this;
      Increment();
      return temp;
}
```

42. Preprocessing occurs
(a) before a program is compiled.
(b) during compilation.
(c) after compilation but before execution.
(d) immediately before execution.

43. `const` variables are preferred to symbolic constants (i.e., `#define`) because
(a) `const` variables require less memory.
(b) symbolic constants can be changed.
(c) `const` variable names are visible to the compiler
(d) `const` variables do not have to be of a specific data type.

44. `#define RECTANGLE_AREA( x, y ) ( ( x ) * ( y ) )` has been defined. Then the line `rectArea = RECTANGLE_AREA( a + 4, b + 7 );` will be expanded to
(a) `rectArea = 11;`

(b) `rectArea = ( a + 4 * b + 7 );`
(c) `rectArea = ( ( a + 4 ) * ( b + 7 ) );`
(d) `RECTANGLE_AREA( a + 4 , b + 7 );`

45. When compared to functions, macros have the disadvantage of
(a) having the overhead of a function call.
(b) increasing program size (if the macros are referenced from many places in the program).
(c) having to fit the macro definition on a single line.
(d) taking only one argument.

46. Large portions of code can be prevented from compiling by

(a)
```
#if 0
      code prevented from compiling
#endif
```
(b)
```
#nodefine
      code prevented from compiling
#endif
```
(c)
```
#if 1
      code prevented from compiling
#endif
```
(d)
```
#ifndef 0
      code prevented from compiling
#endif
```

47. Which of the following is not a valid directive?
(a) `#endif`
(b) `#ifdef`
(c) `#for`
(d) `#elif`

48. Conditional compilation cannot be used to
(a) perform loops.
(b) ignore large blocks of code.
(c) debug programs.
(d) selectively define symbolic constants.

49. Write the output for the following program

(a)
```
int i;
for ( i= 2; i < 10; --i){
      cout << "Here:  " << i << endl;
      i = i + 2;
}
return 0;
```
Output:

```
Here: 1
Here: 2
………
Here: 9
```

50. Write the output for the following program

(b)
```
int sum = 0;
int i = 1, j = 1;
while ( (i = 5) != 10) {
    sum = sum + i;
    if (sum == 20) break;
    j++;
}
cout << i << ":" << j << endl;
```
Output: 5.4

Part II
[Each question is worth 10 points]

Note that you need to attempt only two out of four questions. However, you can get 10 bonus points for one more question.

1. Consider class `CreateAndDestroy` that we discussed in class – it has a *constructor* and a *destructor* function. When an object of this class is created its *constructor* takes two strings as arguments and prints a message. For example, the statement

```
CreateAndDestroy c(1, “(In Example)”);
```

would call its *constructor*, which will simply print out the message

```
          Object 1 constructor runs <In Example>
```

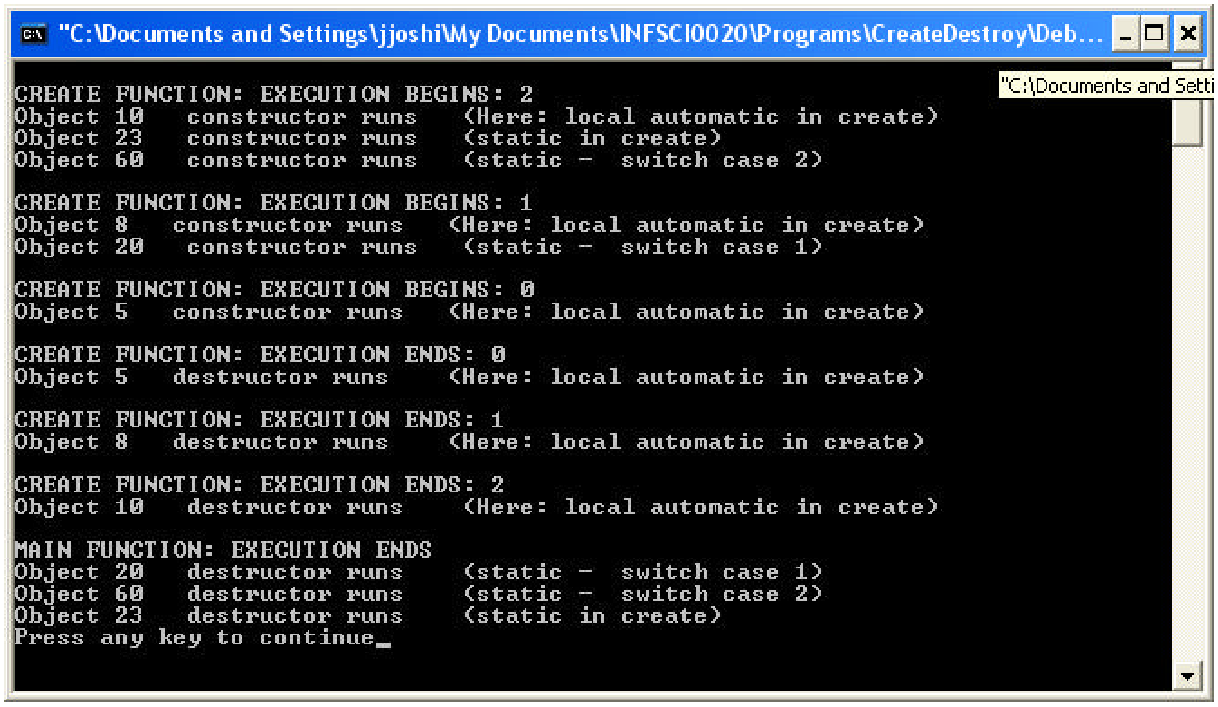
Its *destructor* will, on the other hand will simply print out the message

```
Object 1 constructor runs <In Example>
```

Write the sequence of creation and destruction of objects if the following code is run. Write exactly what it would print.

```
void create(int);   // prototype
int main()
{
   create(2);
   cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
   return 0;
} // end main

// function to create objects
void create(int i)
{  static int j = 1;
   cout << "\nCREATE FUNCTION: EXECUTION BEGINS: " << i << endl;
   CreateAndDestroy x1( i*4 + (++j), "(Here: local automatic in create)" );
   static CreateAndDestroy fifth( i*10 + (++j), "(static in create)" );
   switch(i) {
         case 1:
           {       static CreateAndDestroy x2(i*20, "(static -  switch case 1)");
                   create(i-1);
           }
           break;
         case 2:
           {       static CreateAndDestroy x3(i*30, "(static -  switch case 2)");
                   create(i-1);
           }
         default:
           break;
   }
   cout << "\nCREATE FUNCTION: EXECUTION ENDS: " << i << endl;
}
```


"C:\Documents and Settings\jjoshi\My Documents\INFSCI0020\Programs\CreateDestroy\Deb...
"C:\Documents and Setti

```
CREATE FUNCTION: EXECUTION BEGINS: 2
Object 10   constructor runs    (Here: local automatic in create)
Object 23   constructor runs    (static in create)
Object 60   constructor runs    (static -  switch case 2)

CREATE FUNCTION: EXECUTION BEGINS: 1
Object 8    constructor runs    (Here: local automatic in create)
Object 20   constructor runs    (static -  switch case 1)

CREATE FUNCTION: EXECUTION BEGINS: 0
Object 5    constructor runs    (Here: local automatic in create)

CREATE FUNCTION: EXECUTION ENDS: 0
Object 5    destructor runs     (Here: local automatic in create)

CREATE FUNCTION: EXECUTION ENDS: 1
Object 8    destructor runs     (Here: local automatic in create)

CREATE FUNCTION: EXECUTION ENDS: 2
Object 10   destructor runs     (Here: local automatic in create)

MAIN FUNCTION: EXECUTION ENDS
Object 20   destructor runs     (static -  switch case 1)
Object 60   destructor runs     (static -  switch case 2)
Object 23   destructor runs     (static in create)
Press any key to continue_
```

2. Consider class add and equal functions of the Complex

```
Complex Complex::add( const Complex &right )
{
   Complex temp(realPart + right.realPart, imaginaryPart + right.imaginaryPart );

   return temp;
}
bool Complex::equal( const Complex &right )
{
   return (realPart == right.realPart && imaginaryPart == right.imaginaryPart);

}
```

Assume instead of add and equal functions, you want to use the + and == operators. Define the overloaded + and == operators complex numbers that replace the above functions. For example, you would now be write c = a + b where a, b and c are complex number

3. Let ax^n *where* a is called a coefficient and *n* is an exponent. An example of a term is $2x^3$ (2 is the coefficient and 3 is the exponent). Consider a polynomial as a set of terms added together. For instance

$$10 + 2x^2 + 5x^3$$

is a polynomial where the first term is 10 ($10 = 10x^0$; i.e., coeffient is 10 and exponent is 0), the second term has coefficient 2 and exponent 2; and the third term has coefficient 5 and exponent 3. When you have to add two polynomials you have to add the coefficients of the terms whose exponents are same. For instance, if we want to add the polynomial

$$4x^2 + 3x^3$$

with the one above we would get

$$10 + (2 + 4)\,x^2 + (5 + 3)\,x^3 = 10 + 6\,x^2 + 8\,x^3$$

Develop a class called Polynomial, with one member function that can add two polynomials. You will need to provide the internal representation of each polynomial. Note that for a polynomial is an *array* of terms its and each term can be characterized by two entities – *coefficient* and *exponent*. You need to show the member variables and functions in a class definition, and show how to implement the add member functions

4. Develop a SavingsAccountList class. The class should provide the following capabilities

   a. allows creating a SavingsAccountList object with space for a user specified set of SavingAccount object
   b. allows users to add new SavingAccount objects in the SavingsAccountList object with specified initial balance
   c. allows users to delete the i^{th} savings account from the SavingsAccountList object. The internal representation should maintain the list properly after deletion!
   d. provides a function to prints the balance of the i^{th} savings account upon
   e. provides a function to return the number of currently existing savings accounts

   Define appropriate member variables and functions for the class and show function definitions. You are given the following class definitions for SavingsAccount

```
class SavingsAccount
{
public:
   SavingsAccount( double b ) { savingsBalance = b >= 0 ? b : 0; }
   void calculateMonthlyInterest();
   static void modifyInterestRate( double );
   void printBalance() const;
private:
   double savingsBalance;
   static double annualInterestRate;
}; // end class SavingsAccount
```