Winter 2005 CS 31 (Shinnerl) Practice Midterm

Leave this grid blank, please.

I	II	III	Total

Instructions:

(i) **Do not start yet! Keep the exam closed until instructed otherwise.**

(ii) One two-sided sheet of self-prepared notes only is allowed; otherwise, **no books, notes, or calculating devices**. You are **not** allowed to use your own scratch paper. The back pages of this exam may be used as such. Should you need more paper during the exam, more is available at the front of the room.

(iii) Keep your photo-ID out on your desktop where the proctors can find it. They will come by during the test to check it.

(iv) There is a **strict** time limit of **1 hour and 45 minutes** for this exam. Time warnings will occasionally be posted on the board in front. When time is up, you must stop work **immediately** to avoid a 10% penalty.

(v) This exam consists of three parts. There are 18 multiple-choice questions worth 2 points each in Part I, 5 short-answer questions worth 6 points each in Part II, and 4 short coding questions worth 10 points each in Part III. The total number of points possible is 106, of which 6 points are extra credit. The questions are **not** necessarily in order of increasing difficulty! Answer the easier questions first. Save difficult questions for last.

(vi) Should corrections or hints be deemed necessary, they will be written on the board in front.

(vii) Please provide the information requested below. Scores will be posted online at the `my.ucla` web site in a couple of days.

Name (print) _____

E-mail address: _____

Student ID number: _____

Signature: _____

Discussion Section Number and Time: _____

TA's Name: _____

Part I: Multiple choice questions (2 points each)

Select the single *best answer to each question. Write your answers in the following table.*

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Question 1. What distinguishes the object-oriented paradigm from the procedural paradigm?

(a) The latter is user-centric. The former is programmer-centric.

(b) The latter is programmer-centric. The former is user-centric.

(c) The latter is data-centric. The former is method-centric.

(d) The latter is method-centric. The former is data-centric.

(e) The former is concrete; the latter is abstract.

Question 2. What is the principle of *structured programming*?

(a) Form follows function according to flow chart.

(b) Premature optimization is the root of all evil.

(c) Single entry, single exit.

(d) Omit unnecessary code, but don't be too cryptic.

(e) Useful code is modular and well documented.

Question 3. If `a` and `b` are of type `double`, which of the following expressions is almost always meaningless?
(a) `a = b` (b) `a == b` (c) `a > b` (d) `a < b` (e) `a - b`

Question 4. Which order correctly summarizes the main categories of operator precedence in decreasing order (highest precedence first)?

(a) unary prefix, unary postfix, binary logic, binary arithmetic, assignment

(b) unary postfix, unary prefix, binary arithmetic, binary logic, assignment

(c) binary logic, binary arithmetic, unary prefix, unary postfix, assignment

(d) assignment, binary logic, binary arithmetic, unary postfix, unary prefix

(e) assignment, unary postfix, unary prefix, binary logic, binary arithmetic

Question 5. A simplified representation of the essential features of some object or process is known as

(a) abstraction. (b) information. (c) animation. (d) encapsulation. (e) modularity.

Question 6. What is the difference between *high* and *low* levels of abstraction?

(a) High levels are very detailed and usually difficult to comprehend. Low levels are simple and easy to understand.

(b) Low levels are very detailed and usually difficult to comprehend. High levels are simple and easy to understand.

(c) High levels are common to many programming languages, while low levels are more language specific.

(d) Low levels are common to many programming languages, while high levels are more language specific.

(e) Low levels are unfair, but high levels are not.

Question 7. If a function receives an argument *by value*, then that function

(a) must infer from the value received the type, scope, and lifetime of the associated object.

(b) will automatically reject values that are not in the correct range.

(c) maintains its own storage for that argument separate from any storage that the calling function may have.
(d) shares the calling function's storage for the argument, but uses its own local name for it.
(e) is not allowed to change the value under any circumstances.

Question 8. If a function receives an argument *by reference*, then that function

(a) must infer from the value received the type, scope, and lifetime of the associated object.
(b) will automatically reject values that are not in the correct range.
(c) maintains its own storage for that argument separate from any storage that the calling function may have.
(d) shares the calling function's storage for the argument, but uses its own local name for it.
(e) is not allowed to change the value under any circumstances.

Question 9. How does the compiler match a call to a function with the definition of a function?

(a) by function name and the types and number of arguments.
(b) by function name alone.
(c) by function name and function type.
(d) by function name, function type, and the types and number of arguments.
(e) by process of elimination.

Question 10. If a local variable is not given an initial value in its definition, then

(a) the enclosing source code will not compile.
(b) the program will crash during execution.
(c) the initial value is undefined and unpredictable.
(d) the compiler will set its initial value to zero.
(e) the variable will be assigned a random initial value based on the system clock's current time.

Question 11. What is the difference between a definition and a declaration?

(a) There is no difference.
(b) Every declaration is also a definition, but not vice versa.
(c) A definition binds a name to an object; a declaration introduces a name into a scope.
(d) Declarations are global, but definitions are local.
(e) Only definitions have scope.

Question 12. The scope of a locally declared variable

(a) is unrestricted.
(b) is the enclosing function.
(c) depends on its type.
(d) is the statement in which it appears.
(e) extends from its declaration to the end of the block containing the declaration.

Question 13. The lifetime of an automatic, locally declared variable lasts

(a) for the duration of the program.
(b) until it is killed by the user.
(c) as long as memory for it is available.
(d) as long as it takes to execute all statements in its scope.
(e) as long as the compiler can recall its name.

Question 14. What is the essential distinction between the integral and floating-point types?

(a) Only the integral types are subject to overflow.
(b) There is no essential difference.
(c) Only the floating-point types use exact representations.
(d) Only the integral types use exact representations.
(e) The integral types are symbolic, and the floating-point types are arithmetic.

Question 15. An *expression* is

(a) an uncontrollable control structure.
(b) a statement followed by a semi-colon.
(c) any single line of code in a C++ source file.
(d) a symbolic encapsulation of the entire C++ grammar.
(e) a sequence of symbols that can be reduced to a single value of a specific type.

Question 16. What is *flow of control*?

(a) the smooth coordination of hardware and software.
(b) the build process: compile, link, load, and run.
(c) the order in which the statements of a program are executed.
(d) the program which decides how much time is allocated to each of a collection of concurrent processes.
(e) saying the right thing at the right time.

Question 17. Which programming language mechanisms are indispensable?

(a) polymorphism and encapsulation
(b) `break` and `continue`
(c) repetition and branching
(d) user-defined data types
(e) Nothing is indispensable.

Question 18. Which statement best captures the idea of the *memory pyramid*?

(a) Human memory is fleeting, but magnetic storage lasts forever.
(b) Software reliability is inseparable from hardware reliability.
(c) Document it twice, but debug it only once.
(d) Memory tends to be organized in levels of increasing capacity and access time but decreasing cost.
(e) Advances in storage technology tend to be economically disruptive, rendering previous technology generations obsolete.

Part II: Short Answer Questions (6 points each)

Question 19. Define *modularity* and describe why it is useful.

Question 20. In computer programming, what is a *variable*? List and briefly define all the attributes of a variable in C++.

Question 21. What does the following function do?

```
void puzzle( string s ){
  for (int i = 0; i < s.size(); ++i)
    cout << int(s[i]) << ' ';
}
```

Question 22. What does the following function do?

```
void riddle( int n, int b ){
  if (b > 0 && n > 0){
     riddle(n/b, b);
     cout << n%b;
  }
}
```

Hint: Consider its output when called with specific arguments; e.g., `riddle(5729,10);`

Question 23. Consider the following rough draft of a program for calculating the Grade Point Average (GPA) from a list of grades and corresponding course units entered by the user at run time. Identify the errors in the program, specify whether each is a syntax error or semantic error, and propose corrections. Write your answers in the space to the right of the code.

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

// This program calculates the Grade Point Average from a list of grades
// and their corresponding units inputted by the user at run-time.

int main(){

      int gpa=0, gp=0, units=0, tp=0, tu=0;
      char grade;


      cout << "GPA Program. Enter grades and units \
               one at a time. \n"
           << "Enter q or Q to quit entering grades \
               and calculate GPA.\n";


      do ( grade != 'Q' || 'q' )
      {


          cout << "\nEnter a grade (A through F): ";
          cin  >> grade;

          cout << "\nEnter the units for the grade: ";
          cin  >> units;



          switch(){
          case (grade == 'A' || grade == 'a')
             gp = 4.0*units;
          case (grade == 'B' || grade == 'b')
             gp = 3.0*units;
          case (grade == 'C' || grade == 'c')
             gp = 2.0*units;
          case (grade == 'D' || grade == 'd')
             gp =     units;
          case (grade == 'F' || grade == 'f')
             gp = 0.0;
          default:
          {
              cout << "\nInvalid Grade. Please reenter.\n";
              continue;
          }

          tp += tp + gp;         // tp = running total of grade points
          tu += tu + units;      // tu = running total of units

      }while(true);


      gpa = tp/tu;
      cout << "\n\nGPA = " << gpa << "\n\n";
      return 0;
}
```

Part III: Coding (10 points each)

Partial credit is awarded to partial solutions, including pseudocode. Style and efficiency count, but detailed documentation is not *expected. State any simplifying assumptions that you make.*

Question 24. Write a C++ function `void drawRectangle( int height, int width, char fill )` that prints a filled rectangle to `cout`. The height, width, and fill character of the rectangle are specified by the caller as arguments. For example, the statement `drawRectangle(3,5,'X')` would be displayed as follows.

```
XXXXXXX
XXXXXXX
XXXXXXX
```

Question 25. Write a C++ function `void drawRecRow( int height, int width, string fill )` that prints a *row* of filled rectangles to `cout`. All rectangles in the row have the same shape; hence, only one height and width need be provided by the caller for all the rectangles. The fill character for the ith rectangle in the row is specified by `fill[i-1]`, which may or may not be distinct from those of the other rectangles. The number of rectangles in the row is taken to be the same as the length of the input string of fill characters. For example, the statement `drawRecRow(3,7,"#$%*$");` produces the following output.

```
#######$$$$$$$%%%%%%*******$$$$$$$
#######$$$$$$$%%%%%%*******$$$$$$$
#######$$$$$$$%%%%%%*******$$$$$$$
```

Question 26. Write a C++ function with header `string leftShift( string str )` that cyclically shifts the characters in a string one character to the left, wrapping the leftmost character to the right end of the string. For example, the statement `cout << leftShift("#$%*$");` prints the following output to the screen: `$%*$#`.

Question 27. Use your answers to any of the preceding questions to write a C++ function `void drawQuilt( int height, int width, string fill )` that prints a *sequence* of filled rows of rectangles to `cout`. The number of rows in the sequence is equal to the number of characters in the fill `string`, as is the number of rectangles in each row (the quilt is "square"). As before, all rectangles in the row have the same shape, as specified by the `height` and `width` arguments provided by the caller. The fill characters for the rectangles in each row are taken in sequence from the fill `string` as before, *except* that the fill sequence is shifted cyclically from row to row, one left shift per row. For example, the statement `drawRecRow(3,7,"#$%*$");` produces the following output.

```
#######$$$$$$$%%%%%%%*******$$$$$$$
#######$$$$$$$%%%%%%%*******$$$$$$$
#######$$$$$$$%%%%%%%*******$$$$$$$
$$$$$$$%%%%%%%*******$$$$$$$#######
$$$$$$$%%%%%%%*******$$$$$$$#######
$$$$$$$%%%%%%%*******$$$$$$$#######
%%%%%%%*******$$$$$$$#######$$$$$$$
%%%%%%%*******$$$$$$$#######$$$$$$$
%%%%%%%*******$$$$$$$#######$$$$$$$
*******$$$$$$$#######$$$$$$$%%%%%%%
*******$$$$$$$#######$$$$$$$%%%%%%%
*******$$$$$$$#######$$$$$$$%%%%%%%
$$$$$$$#######$$$$$$$%%%%%%%*******
$$$$$$$#######$$$$$$$%%%%%%%*******
$$$$$$$#######$$$$$$$%%%%%%%*******
```