CS 141 Midterm 2

Spring 2020, Reed and Deitz  
This test was administered online using Blackboard.  
This

|  |  |
| --- | --- |
| Description | This exam will only be available from 6 pm to 7 pm, so be sure you're ready to start on time. |
| Instructions | You are on your honor to follow the instructions given here.  You may lookup information online, but you may not communicate with anyone else and must do all your own work. You may not run code during the test. When only a program segment is given, you can assume it is placed in the context of a program that as much as possible is otherwise correct and includes all declarations and system libraries needed to make it work. |
| Timed Test | This test has a time limit of 1 hour. This test will save and submit automatically when the time expires. Warnings appear when **half the time**, **5 minutes**, **1 minute**, and **30 seconds** remain |
| Multiple Attempts | Not allowed. This test can only be taken once. |
| Force Completion | This test can be saved and resumed at any point until time has expired. The timer will continue to run if you leave the test.   The order of questions was randomized, so when you took the test you likely encountered the questions in a different order.  The point values shown belong to the question *above* those displayed points. |

**QUESTION 1**

Match the code to the C string function it implements.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  char \* Ada( char \*pC, char c)  {      while( \*pC != '\0' && \*pC != c) {          pC++;      }      return pC;  } | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  char \* Pascal(char \* pFirst, char \* pSecond)  {      char \* pTarget = pFirst;      while (\*pTarget != '\0') {          ++pTarget;      }      while (\*pSecond != '\0')      {          \*pTarget = \*pSecond;          ++pTarget;          pSecond++;      }      \*pTarget = '\0';      return pFirst;  } | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  char \* Hopper( char \*pC, char c)  {      char \*pBegin = pC;      while( \*pC != '\0') {          pC++;      }      while( pC != pBegin && \*pC != c) {          pC--;      }      return pC;  } | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ char \* Turing(char \* pFirst, char \* pSecond)  {      char \* pTarget = pFirst;      while (\*pSecond != '\0')      {          \*pTarget = \*pSecond;          ++pTarget;          pSecond++;      }      \*pTarget = '\0';      return pFirst;  } | | |  |  | | --- | --- | | A. | strlen | | B. | strcat | | C. | strrchr | | D. | strcpy | | E. | strncpy | | F. | strchr | | G. | strcmp | | H. | strstr | |

**16 points**

**QUESTION 2**

Consider the following code that attempts to replace blanks in a character array with NULL characters:

char array[10];

for(int i = 0; i <= 10; ++i)

   if(array[i] == ' ')

      array[i] = '\0';

Mark ALL of the following that are a problem with the above code. Do not mark the statements which do not apply or are not an issue.

|  |  |  |
| --- | --- | --- |
|  |  | * array[i] is being used to read values, not write them |
|  |  | * You can't use == in the if statement, rather you have to use strcmp |
|  |  | * Opening and closing curly braces {  } must be added or it won’t run |
|  |  | * The array is improperly declared |
|  |  | * The for loop bounds are incorrect |

**4 points**

**QUESTION 3**

Assume we have the following declarations:

struct Employee{

    char name[ 15];

    int age;

};

struct Employee e1 = {"Jovani", 23};

struct Employee\* pEmployee = &e1;

Mark ALL of the following possible approaches which successfully compile and run, printing out the age for the Employee:

|  |  |  |
| --- | --- | --- |
|  | I. | cout << \*(pEmployee.age) << endl; |
|  | II. | cout << \*pEmployee.age << endl; |
|  | III. | cout << pEmployee->age << endl; |
|  | IV. | cout << \*(&(pEmployee->age)) << endl; |
|  | V. | cout << (\*pEmployee).age << endl; |

**4 points**

**QUESTION 4**

Let's say a class declaration contains *get and set* member functions, as well as other member function definitions.  What approach did we discuss to abstract away some of the details, if the class starts getting too long?

|  |  |  |
| --- | --- | --- |
|  |  | 1. Break up the member functions into smaller pieces. |
|  |  | 1. Define member functions outside of the class, using the class scope resolution operator. |
|  |  | 1. Declare member functions *inline.* |
|  |  | 1. Break up the class into sub-classes, then include parts of other classes. |

**4 points**

**QUESTION 5**

Consider the following code segment. What is the output when function   scope() is called?

int x = 1;   // global

void s2( int y)

{

   cout << x+y;

}

void s1( int y)

{

   x = y++;

   s2( x++);

}

void scope()

{

   int x = 3;

   s1( ++x);

}

**4 points**

**QUESTION 6**

Indentation is meant to clarify what a program does, but must be done correctly.  What is the output from running the program segment shown below?

int num=2; if(num>0) if(num==1) cout<<"bigger ";   
else cout<<"smaller "; cout<<"Done";

|  |  |  |
| --- | --- | --- |
|  |  | No output |
|  |  | 1. bigger |
|  |  | 1. smaller |
|  |  | 1. Done |
|  |  | 1. Bigger Done |
|  |  | 1. Smaller Done |
|  |  | 1. Bigger smaller Done |

**4 points**

**QUESTION 7**

Match each of the terms with the best description.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  | Array | |  | Struct | |  | Pointer | |  | Function | | |  |  | | --- | --- | | A. | Block of code that can be executed by name | | B. | Address of the first element of a collection of objects of the same type | | C. | Way to access a single object even when the original object is not in scope | | D. | Another name for algorithm | | E. | Container for elements of varying types | |

**8 points**

**QUESTION 8**

What is the output from this code?

int values[] = {5,2,3,6,4};

int x = values[0];

int i;

for( i=0; i<5; i++) {

   if( values[ i] > x) {

      x = values[ i];

   }

}

cout << x << endl;

|  |  |  |
| --- | --- | --- |
|  |  | 0 |
|  |  | 1 |
|  |  | 2 |
|  |  | 3 |
|  |  | 4 |
|  |  | 5 |
|  |  | 6 |
|  |  | 7 |

**4 points**

**QUESTION 9**

What is the output of the following code when called with driver()?

void  doIt( int values[], int limit)

{

   int \*pNumber = &values[ 0];

   int x = 0;

   while( pNumber < (values + limit) ) {

      x += \*pNumber;

      pNumber++;

   }

   cout << x << endl;

}

void driver()

{

   int numbers[ ] = { 1,3,2,4};

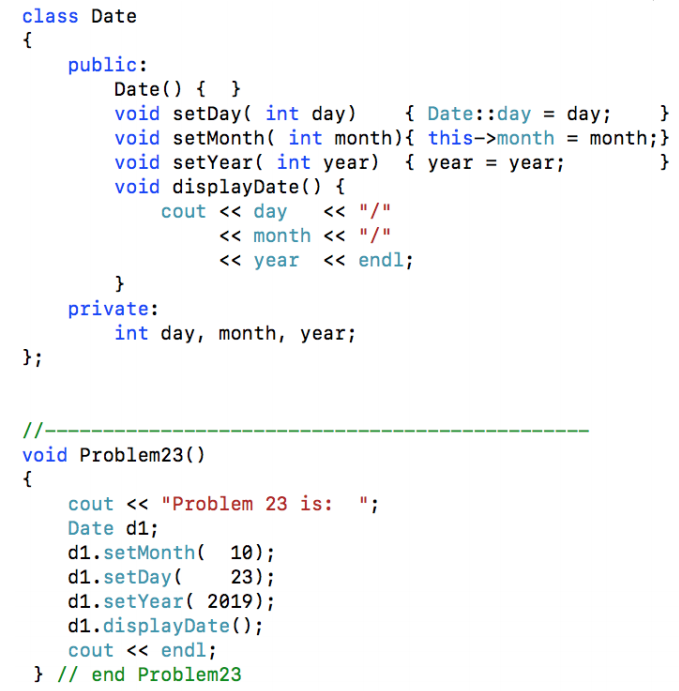
   doIt( numbers, 4);

}

**4 points**

**QUESTION 10**

Consider the code segment shown below. Click on the line that causes a compilation error or that causes the output to not be 10/23/2019.



**4 points**

**QUESTION 11**

Which of the following is the best description of *objects* and *classes*?

|  |  |  |
| --- | --- | --- |
|  |  | 1. *Objects* are a general category, while *classes* are particular instances. |
|  |  | 1. *Classes* are a general category, while *objects* are particular instances. |
|  |  | 1. *Objects* and *classes* are synonyms that describe particular instances. |
|  |  | 1. *Objects* and *classes* are synonyms that describe a general category. |

**4 points**

**QUESTION 12**

Consider using ***bubble sort*** to process the following list of numbers into *ascending* order (smallest to largest), where we make passes right-to-left:

**5 2 3 1 4**

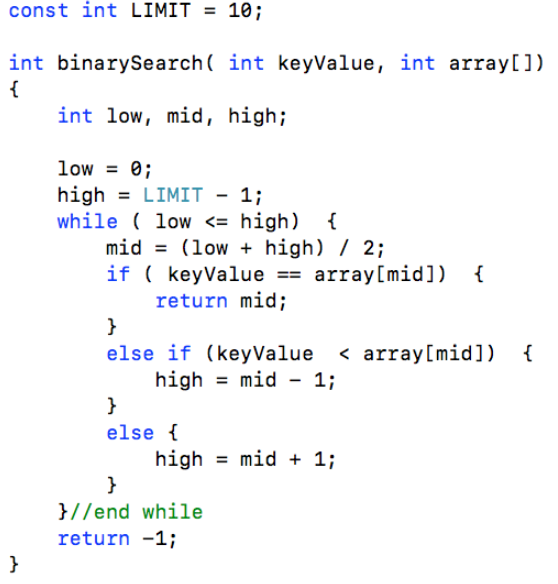
What would be the order of the numbers after making *only the first pass* through the list? Type in your values separated by a single space between each value.



**4 points**

**QUESTION 13**

Consider the following code used to do a binary search that returns the index of keyValue, or -1 if it is not found. Click the line which is ***incorrect*** in this code.



**4 points**

**QUESTION 14**

Consider the following function declaration:

      void setValues( int values[ ]);

Note that there is no number inside the square brackets to specify the array size.  Which of the following statements is the best description of this situation?

|  |  |  |
| --- | --- | --- |
|  |  | 1. This will cause a compiler error and will not run. |
|  |  | 1. This will compile, but will cause a run-time error. |
|  |  | 1. Having no number will allow writing past the end of the array, however writing past the end is prevented when a number value is supplied. |
|  |  | 1. Having no number will allow writing past the end of the array, which is possible regardless of whether or not a number value is supplied. |

**4 points**

**QUESTION 15**

Consider the following code segment. What is the output when function   confuseDriver() is called?

int a=2, b=4, c=6;   // global

void confuse1(int &b, int c)

{

   a = ++b + c;

   b = 2 + a++;

}

void confuse2(int &a, int &b)

{

   b = a + 3;

   c++;

}

void confuseDriver()

{

   int a=1, c=3;

   confuse1( a, b);

   confuse2( b, c);

   cout << a+b+c;

}

**4 points**

**QUESTION 16**

Consider using a *linear search* function to look up a key word which is present in an *unordered* list of words, where we start at the beginning of the list and compare words sequentially.  Assume that after searching for different random key words many times, the ***average number*** of word comparisons required to find each key word is 7.

If this were done instead using *binary search* and the list of words was already sorted, what would be the ***worst-case*** number of word comparisons needed to find some key word which is present?



**4 points**

**QUESTION 17**

When writing code using a *struct*, what is the effect of changing keyword *struct* to *class*?

|  |  |  |
| --- | --- | --- |
|  |  | 1. The program runs correctly, since struct and class are virtually identical. |
|  |  | 1. The program runs, but gives a run-time error when we first access a class instance. |
|  |  | 1. The program may not compile because of a data permission error. |
|  |  | 1. The program will not compile at the point when we try and use a class instance for output. |

**4 points**

**QUESTION 18**

Consider using ***selection sort*** to process the following list of numbers into *ascending* order (smallest to largest), where the first value to be established is the one on the left:

**5 2 3 1 4**

What would be the order of the numbers after making *only the first pass* through the list? Type in your values separated by a single space between each value.



**4 points**

**QUESTION 19**

Consider the three portions of code shown below, where any one of them might be used in a Tic-Tac-Toe game that checks if there are three similar pieces in a row:

I.   if( threeInA\_Row)

        gameDone = true;  
     else

        gameDone = false;

II.  threeInA\_Row ? gameDone = true : gameDone = false;

III.  gameDone = threeInA\_Row ? true : false;

Which of the following statements is true regarding these three portions of code?

|  |
| --- |
| 1. Only one of the three cases would compile and run correctly. |
| 1. The value in gameDone would end up being the same in two of the three cases. |
| 1. The value in gameDone would end up being the same in all three cases. |
| 1. The value in gameDone  would end up being different in all three cases. |

**4 points**

**QUESTION 20**

In C++ we must consider both *assignment* expressions as well as *comparison* expressions. What is the output of this code?

int x = 1;

if( x = 2) {

   cout << "even";

}

else {

   cout << "odd";

}

|  |  |  |
| --- | --- | --- |
|  |  | 1. No output |
|  |  | 1. even |
|  |  | 1. odd |
|  |  | 1. even odd |

**4 points**

**QUESTION 21**

Consider what happens when passing an array to a function.  Mark ALL of the following statements that are true.

|  |  |  |
| --- | --- | --- |
|  |  | 1. Arrays can't be passed to functions, only pointers. |
|  |  | 1. You must catch the array parameter with an ampersand. |
|  |  | 1. The array size must either be a global value or passed as a parameter. |
|  |  | 1. Any changes you make to the array in the function will be reflected in the array in the calling code. |
|  |  | 1. You must pass and catch an array using the same name. |

**4 points**

Bottom of Form