**COSC2347 “C” W/Linux Test 3 Spring 2013 Burris**

Answer any 5 questions. Number your questions on the answer sheets in ascending numeric order from 1 through 7 inclusive. Clearly write “delete” on the answer sheets by the 2 questions you do not wish graded. Leave at least a one-inch margin at the top of every page. You may delete more than one consecutive question on the same page. Do not write on the back of pages. Staple your answer sheets (in ascending numeric order) on top of the test in the upper left hand corner. Write your name in the upper right hand corner of the answer sheets (first page). Turn the stapled bundle over and write your name in the upper right hand corner on the back of the test. All answers must be placed on the answer sheets. No credit will be awarded for work appearing on the test. A five point Road Map Fee (RMF) will be deducted for each instance of failure to follow instructions. You will not receive credit for material I can not read (illegible) or that is obstructed from my view, e.g., by a staple.

Watch your time. Some questions require considerably less time than other questions!

**You may not use C++ on this exam!**

1. The following algorithm sorts an array into ascending order in place. The algorithm is sensitive to the contents being sorted (random, sorted order, reverse sorted order). Evaluate the algorithms performance numerically in terms of the exact number of compares and exchanges required for minimum and maximum run time performance. Approximate the expected number of compares and exchanges considering all data permutations. What will its overall performance be in Big “O” notation for the performance as N 🡪 ∞. **You must support your answer analytically to receive credit.** How do keys with the same value on algorithm entry appear when the sort is complete?

//Sort Grades in ascending order by locating grade[0], then grade[1], etc.

**void AscendingSort( int intArray[ ], int numInt) // CBR and CBV**

**{**

**for(int j = 0; j < numInt - 1; j++ ){** // Reduces list length by 1 each pass.

**for(int k = j + 1; k < numInt; k++ ){** // Moves smallest to top each pass.

**if( grades[ j ] > grades[ k ] ){** //Swap if out of order with current top of list.

**int temp = grades[ j ];**

**grades[ j ] = grades[ k ];**

**grades[ k ] = temp;**

**}**

**}**

**}**

**}**

1. Rewrite the sort algorithm in question one as a pointer sort.
2. Choice (do only one of the following):
3. Write a type 2 recursive function accepting a character string as input and returning the string in reverse order. For example, if the input is “ABCD” the function should return “DCBA.” Use separate parameters for the input and output strings.
4. Write a type 1 recursive function accepting an integer as input and returning the sum of the digits. For example, if the input is the integer 1234 the returned result should be 10.
5. Implement either the A or B option of question 3 but the function must utilize iteration as opposed to recursion to solve the problem.
6. Write a type 1 function to efficiently determine the number of bins containing kites in our warehouse using search method number 1. You must use relative addressing (wareHouse + displacement)! “itemsInTable” is the number of items currently in the table. We typically do not have inventory in all 100 bins. Exhibit a statement as it would appear in the main program to invoke the function. You need not write the entire main program.

**#include<iostream>**

**Sample warehouse contents for itemsInTable = 6.**

|  |  |  |
| --- | --- | --- |
| 0 | kite | 20 |
| 1 | pencils | 16 |
| 2 | paper | 59 |
| 3 | pens | 10 |
| 4 | tops | 25 |
| 5 | kite | 72 |
|  |  |  |
| 99 |  |  |

**using namespace std;**

**#include<string.h>**

**struct bin { char item[20]; int inStock; };**

**int main( ) {**

**struct bin wareHouse[ 100 ] = {**

**{ "kites", 20}, { "pencils", 16},**

**{ "paper", 59}, { "pens", 10},**

**{ "tops", 25}**

**};**

**char itemName[20];**

**int pos;**

**int itemsInTable;**

**int tableSize = sizeof(invTable)/sizeof(struct inventoryItem) - 1;**

1. Compare and contrast search methods 2 and 3. Emphasize why search method 3 is faster than search method 2 and be how much. Your response must be numerical (quantitative) in nature.
2. Write a type 1 function find the total amount of stock in the color “red.” Exhibit a statement as it would appear in the main program to invoke the function. You need not write the entire main program.

**struct bin{ char name[10]; int inStock; char color[10]; };**

**struct bin stock[ 50 ] [ 4 ] [ 3 ];**

1. Write a function in “C” to implement the following. Exhibit an invocation from another function (or main program) to properly invoke your solution.

We require a **recursive** routine to calculate the greatest common divisor (GCD) of two positive integers using Euclid’s Algorithm expressed as GCD(M, N). The algorithm appeared in “Euclid’s Elements” about 300 BC but was apparently discovered up to two hundred years earlier making it one of the oldest know algorithms. The GCD of two positive integers is the largest integer that will divide both of them evenly. For example, the GCD(25, 15) is 5, GCD(15, 9) is 3 and the GCD(7, 15) is 1. Euclid algorithm [GCD(M, N)] obeys the following rules for M > 0 and N >= 0: If N = 0, the GCD is M else the result is GCD(N, M Modulo N). This algorithm is heavily used in cryptography and for calculating the values of many trigonometric functions.

1. A **recursive** routine to perform division of positive integers is required. Fortunately, integer division may be performed as a series of subtractions. Implement DIV(M, N) for M greater than or equal to zero and N greater than zero. Exhibit a sample function invocation from another function (or main program) properly passing all parameters.
2. Write a “C” function to return the surface area and volume of a sphere subject to the restrictions in the following pseudo code specification. The radius of the sphere must be passed by value. The surface area and volume must be returned by reference. You may use **π** from the math library or 3.1417. Exhibit a sample function invocation from another function (or main program) properly passing all parameters.

**void sphere( float byValue radius, float byRef surface, float byRef volume){**

* 1. **surface = 4πr2**
  2. **volume = 4/3πr3**

**// computesthe surface area and volume of the sphere**

**}**

1. Write a function to implement the “strcpy” function available in the standard “C” string library. The function signature is defined as:

void strcpy(char \*copy, char \*original) { /\* code here \*/ }

1. State what the following section of “C” code accomplishes and discuss the general technique used in the code. What role is played by the array “doit?”

**struct entry{ char name[20]; int age; };**

***void mystry( struct entry item[ ], int numEntries){***

int temp, maxPt, doit[ numEntries ];

for(int j = 0; j < numEntries; j++) doit[ j ] = j;

for(int j = (numEntries - 1); j > 0; j-- ){

maxPt = 0;

for(int k = 1; k <= j; k++ )

if( strcmp(item[ doit[k] ].name, item[ doit[maxPt ]].name) > 0 )

maxPt = doit[k];

temp = doit[maxPt];

doit[maxPt] = doit[j];;

doit[j] = temp;

}

for( int k = 0; k < numEntries; k++ )

cout << item[ doit[k] ].name << " " << item[ doit[k] ].age << endl;

**}**

1. Given the following, write a section of code to prompt the user for “itemName.” Print the corresponding price using search method number 3. Assume the desired item appears at most once in the table. You must allow for the case the item is not in the table.

**#include<iostream>**

**using namespace std;**

**#include<string.h>**

**struct inventoryItem { char item[20]; double price; };**

**int main( ) {**

**invTable**

|  |  |  |
| --- | --- | --- |
| 0 | kite | 2.39 |
| 1 | pencils | 16.78 |
| 2 | paper | 0.59 |
| 3 | pens | 10.00 |
| 4 | tops | 25.10 |
| 5 | ????? | ????? |

**struct inventoryItem invTable[ 6 ] = {**

**{ "kites", 2.39}, { "pencils", 16.78},**

**{ "paper", 0.59}, { "pens", 10.00},**

**{ "tops", 25.10}**

**};**

**char itemName[20];**

**int pos;**

**int itemsInTable = 5;**

**int tableSize = sizeof(invTable)/sizeof(struct inventoryItem) - 1;**

1. Prompt the user for the name of an item to be deleted from the table and delete it. You must not leave any holes in the table. For example if paper is deleted from the sample table, the contents of the table would be: kite, pencils, pens, and tops. Remember to decrement the number of items in the table.

**#include<iostream>**

**using namespace std;**

**#include<string.h>**

**struct inventoryItem { char item[20]; double price; };**

**int main( ) {**

**invTable**

|  |  |  |
| --- | --- | --- |
| 0 | kite | 2.39 |
| 1 | pencils | 16.78 |
| 2 | paper | 0.59 |
| 3 | pens | 10.00 |
| 4 | tops | 25.10 |
| 5 | ????? | ????? |

**struct inventoryItem invTable[ 6 ] = {**

**{ "kites", 2.39}, { "pencils", 16.78},**

**{ "paper", 0.59}, { "pens", 10.00},**

**{ "tops", 25.10}**

**};**

**char itemName[20];**

**int pos;**

**int itemsInTable = 5;**

**int tableSize = sizeof(invTable)/sizeof(struct inventoryItem) - 1;**

1. Write a function to efficiently determine the total amount of inventory in stock. You must use relative addressing (invTable + displacement)! For the sample table that would be 20 + 16 + 59 + 10 + 25. We never stock over 100 items.

**#include<iostream>**

**using namespace std;**

**#include<string.h>**

**struct inventoryItem { char item[20]; int inStock; };**

**int main( ) {**

**invTable**

|  |  |  |
| --- | --- | --- |
| 0 | kite | 20 |
| 1 | pencils | 16 |
| 2 | paper | 59 |
| 3 | pens | 10 |
| 4 | tops | 25 |
| 5 | ????? | ????? |

**struct inventoryItem invTable[ 100 ] = {**

**{ "kites", 20}, { "pencils", 16},**

**{ "paper", 59}, { "pens", 10},**

**{ "tops", 25}**

**};**

**char itemName[20];**

**int pos;**

**int itemsInTable = 5;**

**int tableSize = sizeof(invTable)/sizeof(struct inventoryItem) - 1;**

1. Write a section of code to find the total amount of stock in the color “red.”

**struct bin{ char name[10]; int inStock; char color[10]; };**

**struct bin stock[ 50 ] [ 4 ] [ 3 ];**

1. Assume binary tree as defined below pointed to by root. Write a recursive function to traverse the tree starting at the root printing its contents defined as follows:
2. Traverse the right subtree.
3. Visit the node (print its contents)
4. Traverse the left subree.

#include <iostream>

using namespace std;

#define Null 0

struct Node

{ char aChar; struct Node \*Left; struct Node \*Right; };

struct Node \*NewNode(const char Char)

{

struct Node \*Point;

Point = new Node;

Point->aChar = Char;

Point->Left = Null;

Point->Right = Null;

return Point;

}

typedef struct Node \*NodePtr; // \*NodePtr is a synomum (alias) for “struct Node”

struct Node \*Root;

I have completed the first pass making Test 3. It covers all material since the last test. Test 3 require a substantial amount of material from Test 2, especially functions, CBV, CBR, and evaluating algorithm performance (especially sorting algorithms). The test requires the concepts and coding techniques necessary to complete Labs 3, 4, and 5. Those completing the labs prior to the test should enjoy a marked advantage. You have about 1-1/2 weeks till the test to catch up if you are behind. The test require CBV, CBR, strings, arrays, functions, subscripts, relative indexing, sorting techniques, table operations (including random insertion, deletion, and searching), one-dimensional and multiple-dimension tables, sequential search methods (1, 2, and 3), address/pointer arithmetic, structs, and recursion (at least 2 questions) through page 203.

1) What specific function does the following code accomplish? How long does it require to execute in terms of numInt? Express your answer using Big “O” notation for the performance as N 🡪 ∞. **You must support your answer analytically to receive credit.** How do keys with the same value on function entry appear on function exit?

**void Mystery( int intArray[ ], int numInt)**

**{**

**for(int j = 0; j < numInt - 1; j++ ){**

**for(int k = j + 1; k < numInt; k++ ){**

**if( intArray[ j ] >= intArray[ k ] ){** .

**int temp = intArray[ j ];**

**intArray[ j ] = intArray[ k ];**

**intArray[ k ] = temp;**

**}**

**}**

**}**

**}**

2) Discuss the difference between call-by-value and call-by-reference in “C.” Include any limitations on data types that may be passed by value. How does your discussion relate to the Trojan Horse problem.

3) “C” allows two types of functions. Explain each type in detail and how they differ. Your discussion should include logical/physical restrictions on the use of parameters sent to the function and results returned.

4) The function “insertItem” accepts an array of grades, the number of grades currently in the array, and the location the new grade is to be inserted. Write the function code to accomplish the insertion operation. Insertion must be accomplished by moving all grades from the insertion point down one prior to inserting the new grade. Do not destroy any existing grades. Prior to returning to the main program, the number of grades in the array should be incremented by 1 to reflect the array contents is now one larger.

void insertItem( int Grades[ ], int \*numGrades, int insertPoint, int gradeToInsert);

As an example if the grade to be inserted (gradeToInsert) is 16:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Prior to inserting: |  |  |  | After inserting: |  |
|  | Grades |  |  |  | Grades |  |
| 0 | 78 |  |  | 0 | 78 |  |
| 1 | 82 |  |  | 1 | 85 |  |
| 2 | 45 |  |  | 2 | 16 |  |
| 3 | 92 |  |  | 3 | 45 |  |
| 4 | 85 |  |  | 4 | 92 |  |
| 5 |  |  |  | 5 | 85 |  |
| 6 |  |  |  | 6 |  |  |
|  |  |  |  |  |  |  |
|  | numGrades = 5 |  |  |  | If insertPoint = 2, after insertion numGrades = 6 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

5) Given the following declarations, write a section of code to search the table for the desired item. If the item is found, print the price. If the item is not found, print an appropriate message. There are currently 356 items in inventory (locations 0 through 355). The desired item may appear in at most one bin. For full credit, use search method 3. You will receive most of the credit for search method 2, and half credit for search method 1.

struct bin {

char\* item;

double price;

};

struct bin table[ 500 ];

int numItems = 356

char \*desiredItem

6) Explain the difference between search methods one, two, and three. You must clearly delineate the circumstances under which each is better and why.

7) Given the following, what function is Ima Programmer attempting to implement using function hope?

int main( ){

char str1[100];

int ans;

// declarations and code

ans = hope(str1);

}

int hope( char\* pt){ return (pt\* == ‘\0’) ? 0 : 1 + hope(pt +1); }

8) Explain in detail the concept of an abstract data type (ADT). How are ADT’s related to templates? Rewrite “hope” in the above question as a template accepting any intrinsic data type (char, int, long, float, double, byte, short, etceteras).

9) What is printed by the following section of code. Appropriate values are indicated by the comments.

#include <iostream>

using namespace std;

**int main( ){**

**int m, k,**

**\*pt1,**

**\*\*pt2;**

**\*\*\*pt3;**

**k = 1234;**

**m = 56;**

**pt1= &k; // pt1 = 0x0012FF26**

**pt2 = &pt1 // pt2 = 0x0012FF50**

**pt3 = &pt2; // pt3 = 0x0012FF47**

**\*\*pt2 = m;**

**\*pt1 = m + 6;**

**k = k + m;**

**cout << \*\*\*pt3 << endl;**

**}**

10) Write the code to implement the following string function using subscript notation. You may not use relative addressing.

**char \*strcat(char \*first, char \*second);**

11) Write a template class to implement a stack in C++ stored in an array. The user must be able to specify the data type placed in the stack as well as the maximum stack size. Your class must have a destructor to reclaim all storage when the stack object goes out of scope. Finally, create a stack capable of holding 50 double precision numbers using your template.

1. Write a recursive function to multiply two positive integers M and N by successive additions.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

1. Prompt the user for the name of an inventory item then search the following array sequentially utilizing search method number 3. If the item is found, print the unit price. If the item is not found, print an appropriate error message. Your code should work for any legal number of items in the table. The sample table below currently contains 5 items, i.e., numIttems = 5.

**struct inventoryItem { char item[20]; double price; };**

**int seqSchMethod3( struct inventoryItem p1[ ], char key[ ], int numItems );**

**int main( ) {**

**invTable**

|  |  |  |
| --- | --- | --- |
| 0 | Kite | 2.39 |
| 1 | Pencils | 16.78 |
| 2 | Paper | 0.59 |
| 3 | Pens | 10.00 |
| 4 | Tops | 25.10 |
| … | … |  |
| 598 |  |  |
| 599 |  |  |

**struct inventoryItem invTable[ 600 ] = {**

**{ "kites", 2.39}, { "pencils", 16.78},**

**{ "paper", 0.59}, { "pens", 10.00},**

**{ "tops", 25.10}**

**};**

**char itemName[20];**

**int pos;**

**int tableSize = sizeof(invTable)/sizeof(struct inventoryItem) - 1;**

4) The function “deleteItem” accepts an array of grades, the number of grades currently in the array, and the location the new grade is to be deleted. Write the function code to accomplish the deletion operation. Deletion must be accomplished by moving all grades below the deletion point up one. Do not destroy any existing grades. Prior to returning to the main program, the number of grades in the array should be decremented by 1 to reflect the array contents is now one smaller.

void deleteItem( int Grades[ ], int \*numGrades, int deletePoint);

As an example:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Prior to deletion: |  |  |  | After insert: |  |
|  | Grades |  |  |  | Grades |  |
| 0 | 78 |  |  | 0 | 78 |  |
| 1 | 82 |  |  | 1 | 85 |  |
| 2 | 45 |  |  | 2 | 92 |  |
| 3 | 92 |  |  | 3 | 85 |  |
| 4 | 85 |  |  | 4 |  |  |
| 5 |  |  |  | 5 |  |  |
| 6 |  |  |  | 6 |  |  |
|  |  |  |  |  |  |  |
|  | numGrades = 5 |  |  |  | If deletePoint = 2, after deletion numGrades = 4 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Explain the function accomplished by the following function.

**void mystry2(char str1[ ], char str2[ ]) {**

**int k = 0;**

**while ( str1[k] = str2[k] ) k++;**

**}**

|  |  |  |
| --- | --- | --- |
| 0 | Kite | 2.39 |
| 1 | Pencils | 16.78 |
| 2 | Paper | 0.59 |
| 3 | Pens | 10.00 |
| 4 | Tops | 25.10 |
| … | … |  |
| 598 |  |  |
| 599 |  |  |

I have made a second pass constructing the 3rd test. It currently contains 13 questions broken down as follows. I have not yet determined the number of question you must answer (alternately, how many you may delete). The material starts where test 2 left off and goes through the 3 sequential search methods for one dimensional we will discuss this week.

1. Evaluate original algorithms including sorting for run time. You must quantitatively express the minimum and maximum number of operations, e.g., compares and exchanges.
2. Properly utilize relative addressing and subscripts (required in multiple questions).
3. Logically explain and properly utilize call-by-reference (CBR) and call-by-value (CBV) in multiple questions (both scalars and arrays).
4. Discuss and utilize the two types of functions provided by “C.”
5. Random inserting, random deletion, and other operations on single dimension arrays.
6. Sequential search of one dimensional arrays using search methods 1, 2, and 3.
7. Two questions are specifically oriented to strings.
8. Two questions are specifically oriented to recursion.
9. Abstract data types and templates are required for 2 questions.
10. Multiple levels of indirect addressing are utilized.

The following algorithm sorts an array into ascending order in place. Evaluate the algorithms performance numerically in terms of the minimum, maximum, and expected number of compares and exchanges. The algorithm is sensitive to the contents being sorted (random, sorted order, reverse sorted order). What will its overall performance be in Big “O” notation for the performance as N 🡪 ∞. **You must support your answer analytically to receive credit.** How do keys with the same value on algorithm entry appear when the sort is complete?

This algorithm sorts records R1, ... , RN in place with their keys in ascending order.

for J in 2..N loop

I := J -1; K := KJ; R := RJ;

MoveLoop:

loop

if K >= KI then

RI+1 := R; -- This is the final position.

exit MoveLoop;

else

RI+1 := RI; --Search higher.

I := I -1;

if I = 0 then

R1 := R;

exit MoveLoop;

end if;

end if;

end loop MoveLoop;

end loop;