*Fall 2012 Test 1*

*CSE1320 Section 001*

*Tuesday, October 9, 2012 Dr. Tiernan*

Name: Key/Notes Section: 001

Student ID: 1000

Read These Instructions!

1. Fill in your name and the rest of your ten-digit student ID above.

2. This is a **OPEN** book, **OPEN** notes, ***NO CALCULATOR*** test. No digital electronics of any sort are allowed to be used during the test.

3. The test is worth a total of 100 points. The value for each question is given either at the top of that section of questions or in curly braces to the right hand side of the question. There are extra credit questions at the end of the test worth an additional 10 points total.

1. If you do not understand a question, raise your hand and the proctor will come over for you to ask about it. The proctor may or may not answer your question but you should still ask.
2. In questions where I have given you code to use, if you find typos, assume that the code is SUPPOSED to work and just indicate what assumption you made for your answer. This applies in all cases unless the question is specifically about the syntax of the code - in which case there are errors on purpose.
3. I will always try to give partial credit so the more of your work that you show, the more I am able to grade for partial credit if the answer is not entirely correct. It is to your benefit to show your work on the test. Partial work also includes answering part of the question but not all of it.
4. Since you have read the instructions, you will get double extra credit points on the last extra credit question if you mention this instruction after your answer.
5. Don’t get stuck on a question. If you don’t know what to do after thinking about it for a minute and a half, then go on to another question or raise your hand and ask something. You can always go back to questions you skip.
6. You can take the pages of the test apart if desired. Please restaple it neatly when you turn it in.

NO CHEATING!

1.a. Using the values declared below, write a function to do the following:

 Given two parameters, an array of integer values of size MAX and a count of the actual number of values in the array *n* , sum all the positive values in the array and return the sum. Assume that the *n* values in the array are all stored contiguously starting from location 0 and that every value after location *n-1* is set to 0. You may assume that finding a 0 in the array means that you have reached the end of the actual data. Try to be efficient with your code so that it does not execute unnecessarily. {10}

**#define MAX 100**

 /\* some code here and prototype declarations\*/

**int main(void)**

**{**

 **int nums[MAX], n;**

 /\* some more code where values are stored in nums and the count is put in *n* \*/

 /\* FUNCTIONCALL - then your function is called here

**}** /\*end of main \*/

/\* Define your function here. You do NOT have to write the prototype declaration. \*/

int myf(int arr[MAX], int n)

{

int cnt=0, sum = 0;

// three loop conditions work – two are more efficient

// Least efficient: - counts to end of array

// for (cnt = 0, cnt < MAX; cnt++)

// More efficient – only counts to n

// for (cnt =0; cnt < n; cnt++)

// basically equal efficiency – checks for 0 value

while (arr[cnt] != 0)

{ //loop body with if is same for all three loops

if (arr[cnt] > 0)

 sum = sum + arr[cnt];

cnt++; // this line is not needed in the for loops

}

return sum;

}

1.b. Using the declarations and definitions from the previous question, write a code fragment to do the following:

 Given the input array, this code fragment should go through the array and negate the value of every *k*th element in the array. For example, if *k* is 4, then every 4th element starting with element [3] (the fourth element from the beginning of the array) will get negated. Or if *k* is 1, then every element in the array should get negated. Make sure to have a variable that the user can use to specify *k* and add a comment to tell another programmer how they should use this variable. {8}

**int nums[MAX], n; //from prev**

int k, index = -1;

// Tell user that they should enter a value *k* to be used as the repeat length in the array, which means

// *k* will be used to tell the array that every *k*th element should be negated.

// Assume k is given a positive value > 0 somehow

for (index += k; index < MAX; index += k)

{

 nums[index] = nums[index] \* -1;

}

// Another way to do this that many people used was:

for (index = 0; index < MAX; index++)

{

 if (((index+1)%k) == 0)

 nums[index] = -nums[index] ;

}

1.c. Using the declarations and definitions from the previous questions, write a code fragment that would be put at the location of the function call in main (FUNCTIONCALL) in the part *a* question. This code fragment should do the following:

 Write one or more loops that will call the summing function on the array, print the sum, then modify the array elements by negating the value of every element, calling the summing function again, printing the new sum, then negating every second element, calling the summing function, printing the sum, negating every third element, calling the summing function, printing, etc. until you complete the array with every *k-1*th element negated and then summed and printed. If the value of the sum becomes greater than 10,000 at any time, then the loop should stop running, and a message should be printed stating why the loop stopped.

 Please rewrite any previous code fragment that you are using but you do NOT have to rewrite any complete functions that you use. {10}

**int nums[MAX], n; //from prev**

int k=1, index = -1, sum=0;

//Loop to increment through k and check sum > 10,000 and k < (MAX or n)

for (k = 1, ((k < n) && (sum < 10000)) ; k++)

{

 //Call summing function, save return value

 sum = myf(nums, n); //Assume n is number of non-zero elements

 //Print return value

 printf(“\n The sum of positive values in the array on pass %d is %d”, k, sum);

 for (index += k; index < MAX; index += k)

 {

 nums[index] = nums[index] \* -1;

 }

}

if (sum > 10000)

 printf(“\nThe value of the sum was %d which is greater than 10000 so the summing stopped”,sum);

1.d. Write a code fragment that does the same thing as in part *c* but uses a DIFFERENT loop control structure from the one you used in part *c*. {8}

**int nums[MAX], n; //from prev**

int k=1, index = -1, sum=0;

//Loop to increment through k and check sum > 10,000 and k < (MAX or n)

//for (k = 1, ((k < n) && (sum < 10000)) ; k++)

while ((k<n) && (sum < 10000))

{

 //Call summing function, save return value

 sum = myf(nums, n); //Assume n is number of non-zero elements

 //Print return value

 printf(“\n The sum of positive values in the array on pass %d is %d”, k, sum);

 for (index += k; index < MAX; index += k)

 {

 nums[index] = nums[index] \* -1;

 }

 k++;

}

if (sum > 10000)

 printf(“\nThe value of the sum was %d which is greater than 10000 so the summing stopped”,sum);

2. Fill in the blanks to match the operators to the descriptions {1 point each; 6 total}

 Operator Used to:

 E ^ A. temporarily change the type of a variable

 C % B. determine if two values are different

 F || C. find the remainder

 B != D. calculate the integer quotient

 D / E. notate if two bits are the same or different

 A ( ) F. require both choices to be false

3.a. Given *int m = 173510* , convert the value of *m* to binary filling in the spaces below. {6}

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 213 | 212 | 211 | 210 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |

 [HINT: If you show your work, then if you make a mistake, I can still give partial credit. If you just write the answer, it’s either right or wrong.]

 1735

- 1024 2 to the 10

 711

- 512 2 to the 9

 199

- 128 2 to the 7

 71

- 64 2 to the 6

 7

- 4 2 to the 2

 3

- 2 2 to the 1

 1

- 1 2 to the 0

 0

3.b. Perform the operation *~m* and convert the resulting value to hexadecimal. {4}

 The value of ~*m* in hexadecimal is 3938 base 16 The symbol ~ means “bitwise negation”

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |

4. Use the code below to answer the questions that follow.

#include <stdio.h>

#include <string.h>

#define MSTR 12

int main(void)

{

char ray[5][MSTR];

char tom[512];

int w, c=1 ;

for (w=0; ((w<5)&&(c==1)); w++)

 {

 printf(“Please enter a single word longer than 5 char with no spaces or punctuation: “);

 scanf(“%s”, tom);

 if ((strlen(tom) <(MSTR-1)) && (strlen(tom)>5))

 strcpy(ray[w], tom);

 printf(“Do you wish to enter another word? Type 1 to continue or 0 to end: “);

 scanf(“%d”,&c);

 }

}

4.a. Describe in one or two short sentences what the code above is doing. Make sure to indicate how many times things can run in the code. {3}

Reading in and storing from zero to five short words (less than 12 char). The body of the for loop will continue until either the user wants to stop or the array of 5 words is full of valid words.

4.b. If #define FIVE 5 were declared, where could this value be used? Would it r efer to the “same” five in those places? {2}

In array declaration

With for loop counter w

// the loop counter and array declaration go together but the minimum length of the word is a “different” 5

Minimum word length

4.c. What errors does the code check for? Does it notify the user of the errors? {3}

Code checks for minimum and maximum length of word but does not notify the user when a word fails

4.d. Why does string copy use “ray[w]” and not “ray[w][i]” in the strcpy? {2}

ray[w] refers to entire row of array therefore the entire string

ray[w][i] refers to one letter in the string which would not be the correct type.

4.e. What is the maximum number of end-of-string markers could be stored in a char array of size MSTR? What about in a char array of size *MAX* where you do not know the exact size of *MAX*? {3}

12 (MSTR value)

MAX

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| \0 | \0 | \0 | \0 | \0 | \0 | \0 | \0 | \0 | \0 | \0 | \0 |

4.f. What is the maximum number of non-empty strings could be stored in a char array of size MSTR? What about in a char array of size *MAX* where you do not know the exact size of *MAX*? {3}

6 (one char string plus one end-of-string marker)

MAX/2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  | 2 |  | 3 |  | 4 |  | 5 |  | 6 |  |
| a | \0 | e | \0 | i | \0 | o | \0 | u | \0 | y | \0 |

4.g. Starting only at the beginning of the char array of MSTR, how many strings can be stored? What about in a char array of size *MAX* where you do not know the exact size of *MAX*? {2}

1 string

1 string

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  |  |  |  |  |  |  |
| a | v | e | n | u | e | s | \0 | b | o | y | \0 |

4.h. If you wanted to sort the strings in the array, what command(s) could you use and describe how you would use the command. {3}

I would use strcmp to compare the strings and then be able to put them in the order desired. You would compare, then follow the sorting algorithm to put strings into correct locations.

4.i. How are C strings the same as and different from C character arrays? {2}

C strings are arrays of type char which is the same as a character array

C strings can be any length which is the same as a character array

C strings are arrays of type char with an end of string marker stored at the end of the work which is different from a character array

5. Answer the following questions and then code a complete C function to implement the following algorithm:

Given an array *A* of size *NUM* use the following algorithm on the array.

Start at the first element of the array and find the smallest element in the array. Swap the value of that element with the first element of the array.

Starting with the second element, find the smallest element remaining in the array. Swap that value with the value of the second element of the array.

Continue in this fashion until the array only has one element remaining to start with.

Assume that *NUM* is defined as a constant and that *A* is defined in the main routine.

5.a. What control structure do you need in order to “Start at…and find”? Why? {3}

loop (repetition control)

need to look at multiple elements repeatedly to find something

5.b. What control structure do you need in order to “find the smallest”? Why? {3}

Need some sort of selection structure to handle determining what is smaller

5.c. What control structure do you need in order to “Start at first… Start at second…Continue”? Why? {3}

Loop (maybe counted for loop)

Will need to increment first, second, etc.

5.d. What will this algorithm do with array? {4}

This algorithm will sort the array from smallest to largest value

5.e. Write the function to implement the algorithm in the question. {12}

int sortsmall( int A[NUM], int n) // n is the number of elements in the array A

{

int smindex, small;

int index=0, current, smtemp, smtind;

for (current = 0; current < n; current++)

{

 smindex = current;

 small = A[current];

 for(index = current+1; index < n; index++)

 {

 if (A[index] < small)

 {

 smindex = index;

 small = A[index];

 }

 }

 A[smindex] = A[current];

 A[current] = small;

}

Extra Credit questions:

XC1. {4}

int a = 2, b = 3;

short e = 4, f = 5;

long i = 6, j = 7;

signed char o = 8, p = 9;

j = f + p / e – b \* (float) o; The final value of j is -17

j = 5 + (9 / 4) – (3 \* 8) = 5 + 2 – 24 = -17

XC2. What kind of commands are #define and #include and when are they handled during compilation?

 {2}

Preprocessor directives

Handled before other compile steps

XC3. Given the variables and values below, write a printf statement that will produce the output as shown. Assume that the underline marks indicate exact single spaces, i.e. \_ \_ in the output is “ “ (two blanks). {}

float m1 = 845.919, m2 = .0201;

int m3 = 42;

Output should be: \_ \_ 845.91 \_ \_ \_ \_ \_ .02 \_ \_ \_ \_ 42.00

printf(“\Output should be: “); //this line is optional

printf(“%7.2f %8.2f %8.2f”, m1, m2, m3);

XC4. What question should I have asked on this test?

 {Any answer will receive two (2) points}

If Instruction 7 is mentioned HERE after the answer, this question gets 4 points (double credit)