**CS141, summer 2016**

**Final Exam**

**120 Minutes**

**Part I (100 Points)**

**Question 1 (20 points):** What is the output of the program below? Show your work, not just the answer; it doesn’t have to be elaborate, but I want to see your reasoning.

|  |
| --- |
| #include <stdio.h>  int bar(int a) {  int x=a;  x++;  a++;  return 2\*x;  }  int foo(int a, int b) {  int x;  x = b;  a = 2\*b;  b = 3\*x;  return a+b;  }  void main(){  int a, b, c, x;  a = 2;  b = 3;  x = 0;  c = bar(foo(a, b));  printf(“a=%i; b=%i; c=%i; x=%i\n”, a, b, c, x);  return 0;  } |

|  |
| --- |
| **// output here!** |

**Question 2 (25 points):**  Consider the C program below.

**Part I:** What output will this program produce when run?

**Part II:** This program uses for-loops. Write an equivalent

program which does *not* use any for-loops (you may use while loops).

|  |
| --- |
| **#include <stdio.h>**  **#include <stdlib.h>**  **int main() {**  **int i, j,x;**  **x=0;**  **for(i=0; i<8; i = i+2){**  **for(j=1; j<10; j = j\*2){**  **x=x+j;**  **}**  **}**  **printf(“x : %i\n”, x);**  **return 0;**  **}** |

**Question 3 (25 points):** Write a C function which takes an array of integers and an integer n indicating the length of the array reverses the array contents.

For example, the following array: [2, 12, 8, 19, 14]

would be rearranged as: [14, 19, 8, 12, 2]

**Question 4 (30 points):** Write a C function which takes **two** DIM-by-DIM integer arrays (where DIM is some constant) mtxA and mtxB and sets the elements of mtxB such that it represents the *90-degree clockwise rotation of mtxA.*

Example:

mtxA (given):  **3 7 9**

**2 1 3**

**8 7 6**

resulting mtxB:  **8 2 3**

**7 1 7**

**6 3 9**

|  |
| --- |
| **// function template:**  **void rotate(int mtxA[DIM][DIM], int mtxB[DIM][DIM]){**  **}** |

**Part II (100 pts)**

**Question 1 (24 points):** Below is a typedef for of a type called WIDGET followed by three variable declarations.

This is followed by a table of expressions involving these variables. For each expression select one of A..H which best describes the ***type of the expression.!***

|  |  |
| --- | --- |
| **typedef struct {**  **int \*a;**  **int b;**  **char name[10];**  **}WIDGET;** | **// variable declarations**  **WIDGET x;**  **WIDGET \*p;**  **WIDGET things[100];** |

|  |  |
| --- | --- |
| 1. **int** 2. **pointer to int** 3. **WIDGET** 4. **Pointer to WIDGET** | 1. **Pointer to Pointer to WIDGET** 2. **char** 3. **pointer to char** 4. **Invalid Expression** |

|  |  |
| --- | --- |
| **Expression** | **Type: Select from A-H** |
| **&things[20]** |  |
| **x.name[2]** |  |
| **p.b** |  |
| **things[20].a** |  |
| **\*(x.a)** |  |
| **&(x.name[2])** |  |
| **(\*p)->a** |  |
| **&b** |  |
| **\*(p->a)** |  |
| **\*a** |  |
| **&x** |  |
| **\*(things[10].a)** |  |

**Question 2 (26 points):** write a function to ***clone*** an n-element array of integers.

The function takes the array and integer n as parameters; it dynamically allocates an array of the same length and populates it with the values of the given array and returns the allocated array.

**Question 3 (20 points):** You are given an array of n integers representing personality test scores used by a human resources department. The scores are categorized as:

80 or above: YES!

60-79: MAYBE

59 and below: NOOO!

Write a C function which takes such an array, the integer n and reports the number of entries in each category ***via three additional parameters. These three additional arguments are integer pointers.***

**Also give a code segment indicating how to invoke your function**

|  |
| --- |
| // function template  void analyze\_scores(int scores[], int n, int \*n\_yes, int \*n\_maybe, int \*n\_no){  } |
| // code segment showing how to invoke this function  int personality\_scores[100];  int yes, no, maybe;  // code to populate array  **// TODO: call function here using variables above** |

**Question 4 (10 points):** I’m trying to develop an application that uses a stack. My friend says “hey I already have a stack implementation you can use.” She then gives me files stack.c and stack.h.

Great! I look at the files and then start writing my application in a file called my\_app.c

But… I keep getting this annoying message when I try to compile:

“blah blah dereferencing pointer to incomplete type blah blah”

**Briefly** explain what I am probably doing wrong and how it relates to the principles of **Abstract Data Types.**

|  |
| --- |
| *(This question is pretty specific to my style of teaching 141, so you can skip it for now.*  *We'll discuss it in class.)* |

**Question 5 (20 points):**

**5A**: What does foo(8) return where foo is defined as below?

|  |
| --- |
| int foo(int n) {  if(n <= 1)  return 1;  else  return n+foo(n/2) + foo(n/2);  } |

**5B**: see right column

|  |  |
| --- | --- |
| void boo(int n) {  int i, j;  if(n <= 0) return;  for(i=0; i<2; i++) {  for(j=0; j<n; j++) {  printf(“X”);  }  printf(“\n”);  boo(n-1);  }  } | **Question**: what output does boo(3) produce? Hint/suggestion: work bottom-up -- boo(1) first; then boo(2) and finally boo(3). |

**Part III (100 pts)**

**Question 1 (15 points):** below is a C typedef for a linked list node.

Write a function which takes a linked list as a NODE pointer (to the first node in the list) and returns the **sum** of all of the elements/values.

Assume the sum of an empty list is zero.

|  |
| --- |
| typedef struct node {  int val;  struct node \*next;  }NODE;  // you decide on your function parameters, etc. here |

**Question 2 (25 points):** Write a function which compares two given lists “lexicographically” (same as “dictionary order” except here, instead of comparing letters/characters, you are comparing integers).

Examples:

<2, 4, 8> is before <2, 7>

<3, 8, 2, 7> is after <3, 8, 2>

Return values:

A negative integer if listA is “before” listB

Zero if they are identical

A positive integer if listB is “before” listA.

|  |
| --- |
| // Function template  int compare(NODE \*listA, NODE \*listB) {  } |

**Question 3 (60 points):** Using the NODE typedef from question 1 in this section (replicated below), let’s add a “wrapper” struct to give us the basis for a linked-list based implementation of a stack (a stack of integers).

|  |
| --- |
| **typedef struct node {**  **int val;**  **struct node \*next;**  **}NODE;**  **typedef struct {**  **NODE \*top;**  **int size; // Keep this value up-to-date!**  **} STACK;** |

The **top** field, as you may guess, points to the top element in the stack (it is NULL if the stack is empty).

The size field keeps track of how many elements are in the stack so we don’t have to walk the list for the stk\_size function.

Your job: Write the following C functions implementing basic stack operations. Function templates have been provided for you on the following pages. Note that for stk\_create, the return type has been left blank (fill it in).

* **stk\_create**: allocates and initializes an empty stack and returns it to caller (as a pointer to a STACK structure)(10 pts)
* **stk\_destroy**: deallocates an existing stack including all dynamically allocated objects associated with it (think NODEs) (10 pts)
* **stk\_push:** pushes a value onto the top of the stack (15 pts)
* **stk\_pop**: pops and returns the top element on the stack.

***If the stack is empty, you can return an arbitrary value*** (we blame the caller). (15 pts)

* **stk\_size:** returns the number of elements in the stack. This should be a one-liner, but also depends on other functions being correctly implemented (10 pts)

**(function templates below)**

|  |
| --- |
| **// you fill in the return type!**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_ stk\_create(){**  **}** |
| **void stk\_destroy(STACK \*s) {**  **}** |
| **void stk\_push(STACK \*s, int x){**  **}** |
| **// REMEMBER: if stack is empty, return an arbitrary value of**  **// your choosing**  **int stk\_pop(STACK \*s) {**  **}** |
| **// should be a one-liner as long as you’ve kept the size**  **// field up-to-date**  **int stk\_size(STACK \*s) {**  **}** |