CSE 220 – Programming in C

Quiz #2

Spring 2016

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Specify if each of the following statements is True or False (4 pts each).

|  |  |
| --- | --- |
| A function can have multiple return statements in it | T |
| A pointer is a variable used to store the address of another variable | T |
| All arrays with 10 elements require the same amount of memory, regardless of the type of elements stored in the array | F |
| A variable defined inside a block has global scope | F |
| If p and q are two pointers to elements of the same array, adding p and q is a valid and meaningful arithmetic operation | F |

1. Consider the following array. Write statements to do the following: (15 pts)

double a[8] = {0.0, 10.0, 20.0, 30.0, 40.0}, \*p, \*q;

|  |  |
| --- | --- |
| Set p to point to the 1st element of the array | p = a; or p = &a[0]; |
| Set q to point to the element with value 20.0 in the array | q = a + 2; |
| Use p to triple the value stored in a[1] | \*(p+1) = \*(p+1)\*3 |
| Let q point to the third element that follows it | q = q + 3 |
| Print the address of p | printf(“%p”, &p); |
| Using p and q, print the number of elements separating the values they point to | printf(“%d”, p - q); |
| Write an if statement that prints “YES” if p and q point to the same element and “NO” otherwise  if ( p == q)  printf(“YES”);  else  printf(“NO”); | |

1. Write a recursive function that takes an array of integers and returns 1 if all its elements are equal and 0 otherwise. (15 pts):

int allEqual(int array[ ], int n) {

//If the array has 1 or 0 elements, all elements are equal

If (n == 0 || n == 1)

return 1;

//If the last two elements are different, then not all equal

if (array[n-1] != array[n-2])

return 0;

else

//Check if all are equal starting from second to last

return allEqual(array, n-1);

}

1. What is the output of the following c program (21 pts).

#include <stdio.h>

int a;

void increase(void) {

a = a + 2;

printf(“I. a = %d\n”, a);

}

int decrease(int a) {

int b = a / 2;

printf(“D. b = %d\n”, b);

}

void swap(int a, int \*b) {

int temp = a;

a = \*b;

\*b = temp;

}

int main() {

a = 5;

int b = 20;

if (b > 15) {

int a = 53;

decrease(b);

increase();

printf(“B. a = %d\n”, a);

}

printf(“M1. a = %d, b = %d\n”, a, b);

swap(a, &b);

printf(“M2. a = %d, b = %d\n”, a, b);

return 0;

}

D. b = 10

I. a = 7

B. a = 53

M1. a = 7, b = 20

M2. a = 7, b = 7

1. The function innerProduct takes as input two pointers to int to represent arrays and their size and returns the inner product of the arrays. The inner product of two arrays a and b of size n is defined as a[0]\*b[0] + a[1]\*b[1] + … + a[n-1]\*b[n-1]. Finish the following program (29 pts)

/\* include any libraries needed \*/ (3)

#include<stdio.h>

#include<stdlib.h>

/\* declare the function innerProduct (prototype only) \*/ (3)

int innerProduct(int \*a, int \*b, int n);

int main(){

/\* declare two arrays of type integer of size 25 \*/ (3)

int N = 25;

int arrayA[N], arrayB[N];

/\* Initialize the arrays to random numbers \*/ (4)

int idx;

for (idx = 0; idx < N; idx++) {

arrayA[idx] = rand();

arrayB[idx] = rand();

}

/\* Call innerProduct on your array and print the result \*/ (5)

int result = innerProduct(arrayA, arrayB, N);

printf(“Result: %d\n”, result);

}

/\* define the function innerProduct \*/ (11)

int innerProduct(int \*a, int \*b, int n) {

int result = 0, idx;

for (idx = 0; idx < n; idx++) {

result += \*(a+idx) \* (\*(b+idx));

}

return result;

}