A1

Review Qs: Refer to handout

1. What value is at address 0x16fdceff0?

2. What value is at address 0x16fdceff8? (note: sizeof(double)=8)

3. What is surprising about res1 and res2?

1.
- 8-byte double representing 1.3
  1.3
- the first byte of the 8 byte double for 1.3
- the locally-declared array vec1

2. double for 4.2; the first byte of the double 4.2; the second element of vec1

3. res1 and res2 print as the same address
   res1[0] and res2[0] are both 9333
   (expect res1[0] to be 1.3+1.5=1.8)

A H

```
1 #include <stdio.h>
2
3 // vector_sum: takes two same-length vectors (double[])
4 // adds them together component-wise in a new array
5 // vector_sum({ 1.2, 3.4 }, {-1.0, 3.6 }) => { 0.2, 7.0 }
6 // Assume the vectors have the same length
7
8 // Q: What happens if double[] is used as a return type?
9 // double[] vector_sum(double vec1[], double vec2[]);
```

syntax error

```
10
11 // Q: What about using double* as return type?
12 // double* vector_sum(double vec1[], double vec2[])
```

↳ double res[. . .] don't know length!

```
13
14 // Pass in length as an argument. Maybe now we've got it!
15 double* vector_sum(double* v1, double* v2, int len) {
16   double res[len];
17   printf("v1@%p : %p\tv2@%p : %p\tres: %p\n",
18            &v1,  v1,   &v2,  v2,     res);
19   for(int i = 0; i < len; i += 1) { res[i] = v1[i] + v2[i]; }
20   return res;
```

X the warning should be an error

```
21 }
22 int main() {
23   double vec1[] = { 1.3, 4.2 }, vec2[] = { 1.5, -1 };
24   double* res1 = vector_sum(vec1, vec2, 2);
25
26   double vec3[] = { 333, 222 }, vec4[] = { 9000, 1000 };
27   double* res2 = vector_sum(vec3, vec4, 2);
28
29   printf("res1[0]: %f\t res2[0]: %f\n", res1[0], res2[0]);
30
31   printf("vec1: %p\n", vec1);
32   printf("vec2: %p\n", vec2);
33   printf("vec3: %p\n", vec3);
34   printf("vec4: %p\n", vec4);
35   printf("res1: %p\n", res1);
36   printf("res2: %p\n", res1);
37 }
```

&v1 v1 &v2 v2

```
$ gcc vector_sum.c -o vector_sum
vector_sum.c:22:10: warning: address of stack memory associated with
    local variable 'res' returned [-Wreturn-stack-address]
  20 |   return res;
$ ./vector_sum
v1@0x16fdcef50 : 0x16fdceff0 v2@0x16fdcef48 : 0x16fdcefe0 res: 0x16fdcef00
v1@0x16fdcef50 : 0x16fdcefd0 v2@0x16fdcef48 : 0x16fdcefc0 res: 0x16fdcef00
res1[0]: 9333.000000   res2[0]: 9333.000000
vec1: 0x16fdceff0
vec2: 0x16fdcefe0
vec3: 0x16fdcefd0
vec4: 0x16fdcefc0
res1: 0x16fdcef00
res2: 0x16fdcef00
```

Variable/Role	Address	Data
res	0x...00	~~2.8~~ 9333
	0x...08	~~3.2~~ 1222
	0x...10	
	0x...18	
	0x...20	
	0x...28	
	0x...30	
	0x...38	
	0x...40	
v2	0x...48	~~0x16fdcefe0~~ 0x...c0
v1	0x...50	~~0x16fdceff0~~ 0x...d0
	0x...58	
	0x...60	
	0x...68	
	0x...70	
	0x...78	
	0x...80	
	0x...88	
	0x...90	
	0x...98	
	0x...A0	
	0x...A8	
res1	0x...B0	0x16fdcef00
res2	0x...B8	0x16fdcef00
vec4	0x...C0	← 9000 →
	0x...C8	← 1000 →
vec3	0x...D0	← 333 →
	0x...D8	← 222 →
vec2	0x...E0	← 1.5 →
	0x...E8	← -1 →
vec1	0x...F0	← 1.3 →
	0x...F8	← 4.2 →

stack ↑

vector sum stack frame

main stack frame

sizeof (x) => compile-time operation, <u>not</u> a function A2

x could be

- a type : gives the # of bytes to store 1 of that type

  int32_t = 4   double* = 8
  double = 8   char* = 8
  char = 1   int32_t* = 8

- an expression (for example, a variable)
  gives # of bytes to store 1 of the type of that expression/variable

  char c = 'a';   int32_t i = 22;
  sizeof(c) = 1   sizeof(i) = 4

- an array variable
  gives total # of bytes for the array declaration

  char c[9];   double ns[5];
  sizeof(c) = 9   sizeof(ns) = 40

  char a[] = "abc";
  sizeof(a) = 4

---

v_s (double* v1; . . .)

v1[3] => look up 8 bytes at v1 + 24

(char* s)

s[3] look up 1 byte at s + 3

(int* ns)

ns[3] look up 4 bytes at ns + 12

sizeof(double)   3* sizeof(double)
(char)   (char)
(int)   (int)