.1

א. הפלט יהיה:

2.6, undefined, undefined, undefined, undefined,

מכיוון שכאשר מערך מועבר כפרמטר לפונקציה, מועבר בפועל רק מצביע למערך, ואז נקבל:

sizeof(vin) / sizeof(vin[0]) = sizeof(void*) / sizeof(double) = 8 / 8 = 1

לכן הפונקציה תכניס ערך רק באיבר הראשון של הוקטור.

בשאר האיברים יש תוכן לא מוגדר, זבל שנשאר שם מריצה קודמת.

ב.

```
void ScalarMultiplication(const double vin[], double scalar, double vout[], size_t dimension)
{
	size_t i;
	for (size_t i=0; i<dimension; ++i)
	{
		vout[i] = scalar * vin[i];
	}
}
...
ScalarMultiplication(vin, scalar, vout, sizeof(vin)/sizeof(vin[0]));
...
```

2. השורה

SAFE_MALLOC(malloc(sizeof(int)),p);

תתורגם על ידי ה pre-processor ל:

if((malloc(sizeof(int)))!=((void *)0)) {(p)=(malloc(sizeof(int)));} else {exit(-1);};

כלומר, תהיה דליפת זיכרון מכיוון שאם ההקצאה מצליחה אנו מבקשים הקצאה נוספת.

בנוסף, יש לשים לב כי אין בדיקה של ההקצאה השנייה.

3

מספר שורה	כתובת	מיקום
1	&(a._arr[1]) (22)	מחסנית
2	&(a._arr[2]) (22)	לא מוגדר
3	&bptr (26)	מחסנית
4	bptr (26)	ערימה דינמית
5	bptr->_name (26)	איזור הקוד
6	bptr2 (27)	איזור הקוד
7	bptr->_name (34)	ערימה דינמית
8	&B_new_called (13)	גלובלי

ב.

לפני ה main:

```
void B_free(B* bptr) {
        free(bptr);
}
```

אחרי שורה 36:

```
        B_free(bptr);
        free(newstr);
```

<u>שאלה 3 א</u>

```
void scalar_binary_op(void* out, const void* in1, const void* in2, size_t nmemb, size_t size,
void (*op)(void*,const void *, const void *)) {
        size_t i;
        for (i=0; i<nmemb; ++i, out+=size, in1+=size, in2+=size) {
                op(out, in1,in2);
        }
}
```

שאלה 3 ב

```
void int_mult_op(void* out, const void * ip1, const void * ip2) {
	const int i1= *((const int*)ip1);
	const int i2= *((const int*)ip2);
	int* outiptr= (int*)out;
	*outiptr= i1*i2;
}

TEST(int_mult_op, test1) {
	int i1= 3;
	int i2= 8;
	int res=0;

	int_mult_op(&res, &i1, &i2);
	EXPECT_EQ(res,24);
}
TEST(scalar_binary_op, test1) {
	int in1[]=	{2,   3, -1, 0};
	int in2[]=	{9,   8,  6, 99};
	int expected_out[]= {18, 24, -6, 0};
	int out[sizeof(in1)/sizeof(*in1)];

	scalar_binary_op(out, in1, in2, sizeof(in1)/sizeof(*in1), sizeof(*in1), &int_mult_op);
	size_t i;
	for (i=0; i<sizeof(in1)/sizeof(*in1); ++i) {
		EXPECT_EQ(expected_out[i],out[i]);
	}
}
```