Memory management in C

1.) List all memory resources used by each of the following statements found inside a C function definition.

i. char *info = "hello";
ii. int numbers[4];
iii. char *data = (char *) malloc(1024);
iv. char **bigdata = (char **) malloc(1024*sizeof(char *));

2.) Imagine you are writing an algorithm that uses a binary tree data structure with nodes defined as below. Since space for all the contents of this data structure was allocated on the heap, you need to de-allocate all of the associated memory; write an implementation of free_tree that successfully does this. You may assume that the tree is of reasonably shallow depth.

```c
struct treenode {
    struct treenode *left_child;
    struct treenode *right_child;
    char *value;
};

void free_tree(struct treenode *root) {
}
```

3.) Part I: Can you find and comment the memory management bugs below? Part II: Fix them to the right.

```c
#define LEN 64

int *do_things(int *data) {
    int vector[LEN];
    int *tmp = vector;
    int *values = (int *) malloc(LEN);
    int *result = (int *) malloc(LEN);
    // do math, populating result
    return result;
}
```

```c
#define LEN 64

int *do_things(int *data) {
    int vector[LEN];
    int *tmp = vector;
    int *values = (int *) malloc(LEN); // Fix allocation here
    int *result = (int *) malloc(LEN);
    // do math, populating result
    return result;
}
```
Intro to MIPS

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add dst, src0, src1</td>
<td>add $s0, $s1, $s2</td>
</tr>
<tr>
<td>add immediate</td>
<td>addi dst, src0, immedi</td>
<td>addi $s0, $s1, 12</td>
</tr>
<tr>
<td>shift left logical</td>
<td>sll dst, src, shamt</td>
<td>sll $t0, $s0, 4</td>
</tr>
<tr>
<td>load word</td>
<td>lw dst, offset(bAddr)</td>
<td>lw $t0, 4($s0)</td>
</tr>
<tr>
<td>store word</td>
<td>sw src, offset(bAddr)</td>
<td>sw $t0, 4($s0)</td>
</tr>
<tr>
<td>branch if not equal</td>
<td>bne src0, src1, brAddr</td>
<td>bne $t0, $t1, notEq</td>
</tr>
<tr>
<td>branch if equal</td>
<td>beq src0, src1, brAddr</td>
<td>beq $t0, $t1, Eq</td>
</tr>
<tr>
<td>jump unconditional</td>
<td>j jumpAddr</td>
<td>j jumpWhenDone</td>
</tr>
<tr>
<td>jump register</td>
<td>jr reg</td>
<td>jr $ra</td>
</tr>
</tbody>
</table>

Translate each of the following C-code snippets into MIPS assembly. Use up to eight instructions for each segment, but limit the instructions used to those listed in the table above.

1. Assume a is held in $s0, b is held in $s1, c is held in $s2, and z is held in $s3.

```c
int a=4, b=5, c=6, z;
z = a+b+c+10;
```

2. Assume $s0 holds $p after `int *p = (int *) malloc(3*sizeof(int))` and $s1 holds a.

```c
p[0] = 0;
int a = 2;
p[1] = a;
p[a] = a;
```

3. Assume $s0 holds a and $s1 holds b.

```c
int a = 5, b = 10;
if (a + a == b) {
    a = 0;
} else {
    b = a - 1;
}
```

Interpret the following MIPS assembly code and provide a written explanation of what it does.

```mips
addi $s0, $0, 0
addi $s1, $0, 1
addi $t0, $0, 30
loop: beq $s0, $t0, done
sll $s1, $s1, 1
addi $s0, $s0, 1
j loop
```

Answer:

```
addi $s0, $0, 0
addi $s1, $0, 1
addi $t0, $0, 30
loop: beq $s0, $t0, done
sll $s1, $s1, 1
addi $s0, $s0, 1
j loop
```

```
done: # done!
```