1. Given the following two functions which might appear to be performing the same task, write one implementation for `f()` such that they do not produce the same value, and another where the optimized second version would be valid because they would produce the same results.

```
int f(int);

int func1(int x) {
  return f(x) + f(x) + f(x) + f(x);
}
```

```
int f(int);

int func2(int x) {
  return 4 * f(x);
}
```

2. Write the body of the second loop so that the two loops would have equivalent effects.

```
for (i= 0; i < n; i++)
   c[i]= a[i] + b[i];
```

```
for (i= 0; i < n; i += 4)
```

3. The following loop will reverse all of the elements of an array:

```
for (i= 0; i < SZ / 2; i++) {
  temp= arr[i];
  arr[i]= arr[SZ - i - 1];
  arr[SZ - i - 1]= temp;
}
```

Is there any opportunity for optimization in this loop? If so, rewrite the loop to incorporate as many optimizations which a compiler might be able to make as you can see.

4. Assume there are three variants of a program which perform the same task, but differ in their efficiency as follows:
   - Version 1 takes $60 + 35n$ time.
   - Version 2 takes $135 + 4n$ time.
   - Version 3 takes $157 + 1.25n$ time.

   If these functions measure the programs' running times (in clock cycles), for what values of n would each version be the fastest (n is an integer)?

5. Consider the following functions:

```
int min(int x, int y) {
  return x < y ? x : y;
}

int max(int x, int y) {
  return x < y ? y: x;
}
```

```
void incr(int *xp, int v) {
  *xp += v;
}

int square(int x) {
  return x * x;
}
```

The following code segments call these functions. If `x` is 10 and `y` is 100, fill in the following table indicating the number of times each of the four functions ends up being called in code segments (a), (b) and (c).

a.
```
for (i= min(x, y); i < max(x, y); incr(&i, 1))
    t += square(i);
```

b.
```
for (i= max(x, y) - 1; i >= min(x, y); incr(&i, -1))
    t += square(i);
```

c.
```
int low= min(x, y);
int high= max(x, y);
for (i= low; i < high; incr(&i, 1))
  t += square(i);
```

code	`min`	`max`	`incr`	`square`
(a)				
(b)				
(c)				