Q1) Using the vector data structure from the book:

```
typedef int data_t;
typedef struct {
       long len;
       data_t *data;
       } vec_rec, *vec_ptr;

/* Return length of vector */
long vec_length(vec_ptr v) {
       return v->len;
       }

int get_vec_element(vec_ptr v, long index, data_t *dest){
       if (index < 0 || index >= v->len)
               return 0;
       *dest = v->data[index];
       return 1;
       }

/*Return pointer to start of vector data*/
data_t *get_vec_start(vec_ptr v) {
       return v->data;
       }
```

The following is a function that computes the inner product of two vectors with equal length.

A) (45 points) Apply the techniques for optimization of code and provide the new revised function. This includes:

   1. Eliminating loop inefficiencies (code motion)
   2. Reducing procedure calls
   3. Eliminating unneeded memory references

B) (45 points) Further revise your loop from the result in A by applying loop unrolling and provide the inside of the loop. Unroll the loop 6 times using 2 accumulators. Account for remainders if necessary.

C) (10 points) Provide the revised function after the changes in A and B.

```
/* Inner Product. Unoptimized version */
void inner_product(vec_ptr u, vec_ptr v, data_t *dest) {
long i;
long length = vec_length(u);
data_t udata;
data_t vdata;
data_t sum = (data_t) 0;
for(i=0; i<vec_length(u); i++) {
       vec_ptr a = u;
       vec_ptr b = v;
       get_vec_element(a,i,&udata);
       get_vec_element(b,i,&vdata);
       *dest = *dest + udata + vdata;
       }
}
```

Part A – Code with optimizations

```
typedef int data_t;

typedef struct {
    long len;
    data_t *data;
} vec_rec, *vec_ptr;

void inner_product(vec_ptr u, vec_ptr v, data_t *dest) {
    long i;
    long length = *((long*)u);
    data_t *udata = u->data;
    data_t *vdata = v->data;
    data_t utmp, vtmp, sum = 0;
    for(i=0; i<length; i++) {
        utmp = udata[i];
        vtmp = vdata[i];
        sum += utmp + vtmp;
    }
    *dest = sum;
}
```

Part B – Loop unrolled 6 times, using 2 accumulators

```
typedef int data_t;

typedef struct {
    long len;
    data_t *data;
} vec_rec, *vec_ptr;

void inner_product(vec_ptr u, vec_ptr v, data_t *dest) {
    long i;
    long length = *((long*)u);
    long lim = length - length % 6;
    data_t *udata = u->data;
    data_t *vdata = v->data;
    data_t utmp1, utmp2, vtmp1, vtmp2, sum = 0;
    for(i=0; i<lim; i+=6) {
        utmp1 = udata[i];
        vtmp1 = vdata[i];
        sum += utmp1 + vtmp1;
        utmp2 = udata[i+1];
        vtmp2 = vdata[i+1];
        sum += utmp2 + vtmp2;

        utmp1 = udata[i+2];
        vtmp1 = vdata[i+2];
        sum += utmp1 + vtmp1;
        utmp2 = udata[i+3];
        vtmp2 = vdata[i+3];
        sum += utmp2 + vtmp2;

        utmp1 = udata[i+4];
        vtmp1 = vdata[i+4];
        sum += utmp1 + vtmp1;
        utmp2 = udata[i+5];
        vtmp2 = vdata[i+5];
        sum += utmp2 + vtmp2;
    }

    for (; i<length; i++) {
        utmp1 = udata[i];
        vtmp1 = vdata[i];
        sum += utmp1 + vtmp1;
    }
    *dest = sum;
}
```

Part C – Fully revised function

```
typedef int data_t;

typedef struct {
    long len;
    data_t *data;
} vec_rec, *vec_ptr;

void inner_product(vec_ptr u, vec_ptr v, data_t *dest) {
    long i;
    long length = *((long*)u);
    long lim = length - length % 6;
    data_t *udata = u->data;
    data_t *vdata = v->data;
    data_t utmp1, utmp2, vtmp1, vtmp2, sum = 0;
    for(i=0; i<lim; i+=6) {
        utmp1 = udata[i];
        vtmp1 = vdata[i];
        sum += utmp1 + vtmp1;
        utmp2 = udata[i+1];
        vtmp2 = vdata[i+1];
        sum += utmp2 + vtmp2;

        utmp1 = udata[i+2];
        vtmp1 = vdata[i+2];
        sum += utmp1 + vtmp1;
        utmp2 = udata[i+3];
        vtmp2 = vdata[i+3];
        sum += utmp2 + vtmp2;

        utmp1 = udata[i+4];
        vtmp1 = vdata[i+4];
        sum += utmp1 + vtmp1;
        utmp2 = udata[i+5];
        vtmp2 = vdata[i+5];
        sum += utmp2 + vtmp2;
    }

    for (; i<length; i++) {
        utmp1 = udata[i];
        vtmp1 = vdata[i];
        sum += utmp1 + vtmp1;
    }
    *dest = sum;
}
```