Name:- Sumit Kumar Yadav
Roll No.:- 18CS30042 || Assignment-7

(1) C code:

```
int a[10], c[10];
int n=10, i;
for (i=0; i<n; i++) {
        if (a[i] % 2 == 0)
            c[i] = 0
        else
            c[i] = 1
}
return;
```

Optimization of Three Address Code using peep-hole optimization :-

```
100 : t1 = 10 xxx
100 : 101 : n = 10 ⇦ def. use
      102 : t2 = 0 xxx
101 : 103 : i = 0 ⇦ def. use
102 : 104 : if i<n goto 109
103 : 105 : goto 125
      106 : t3 = i xxx
104 : 107 : i = i+1 ⇦ def. use
105 : 108 : goto 104
106 : 109 : t4 = 4*i      // strength reduction
107 : 110 : t5 = a[t4]
108 : 111 : t6 = t5 % 2   // strength reduction
      112 : d7 = 0 x xx
109 : 113 : if t6 != 0 goto 120 // jump over jmp & def use
      114 : goto 115 xx x
110 : 115 : t8 = 4*i      // strength reduction
```

```
111 : 116 : t9 = c + t8
      117 : t10 = 0 x x x
112 : 118 : *t9 = 0        ⇐ def. use
113 : 119 : goto 106
114 : 120 : t11 = 4 * i    // strength reduction
115 : 121 : t12 = c + t11
      122 : t13 = 1  x x x
116 : 123 : *t12 = 1       ⇐ def. use
117 : 124 : goto 106
118 : 125 : return
```

On removal and Reduction :-

```
100 : n = 10
101 : i = 0
102 : if i < n goto 106
103 : goto 118
104 : i = i + 1
105 : goto 102
106 : t4 = i << 2
107 : t5 = a[t4]
108 : t6 = t5 & 1
109 : if t6 != 0 goto 114
110 : t8 = i << 2
111 : t9 = c + t8
112 : *t9 = 0
113 : goto 104
114 : t11 = i << 2
115 : t12 = c + t11
116 : *t12 = 1
117 : goto 104
118 : return
```

(2) (a) code, which prepares the stack and registers for use within the function :-

```
push  ebp
mov   ebp , esp
sub   esp , N
push  esi
```

(b) code, which restores the stack and registers to the state they were in before the function was called.

```
pop   esi
mov   esp , ebp
pop   ebp
ret   0
```

(3) list of live variables :- (liveness analysis)

```
000 :
001 : count = 0                          // a,n
002 : i = 0                              // a,n, count
003 : L0 : if i<n goto L2                // a,n, count, i
004 :  goto L3                           // a,n, count, i
005 : L1 : i = i+1                       // a, n, count, i
006 :  goto L0                           // a,n, count, i
007 : L2 : t0 = 4*i                      // a,n, count, i
008 : t1 = a[t0]                         // a, n, count, i, t0
009 : t2 = t1 % 2                        // a,n, count, i, t0, t1
010 : if t2 != 0 goto L1                 // a, n , count, i, t1, t2
011 : count = count + 1                  // a,n, count, i, t2
012 : goto L1                            // a,n, count, i
013 : L3 : return count                  // a, n, count, i
                                         // count
```

Interval Graph :-