Halmstad University
Center for Research on Embedded Systems

Solutions to the Embedded Systems Programming Examination, April 2012

1.a.

```
void  set(unsigned int * port, unsigned int mask){
  *port = *port | mask;
}
void  clear (unsigned int * port, unsigned int mask){
  *port = *port & ~mask;
}
```

1.b.

```
set(PORT,0x15)
```

2.a.

We are forced to choose an order to wait for input doing busy waiting on a given input source and risking to miss input from other sources. The CPU is executing all the time, even if there is nothing to be done, just testing registers waiting for input.

2.b.

One way of reorganizing the program is to destroy the read functions and do the busy waiting in the main loop:

```
main(){
 struct Params params;
 struct Packet packet;
 int dist, signal;
 while(1){
  if (SONAR_STATUS & READY == 1){
      dist = SONAR_DATA;
      control(dist, &signal, &params);
      servo_write(signal);
   }
   if (RADIO_STATUS & READY == 1){
      pkt->v1 = RADIO_DATA1;
      ...
      pkt->vn = RADIO_DATAn;
      decode(&packet,&params);
    }
 }
}
```

One remaining problem is that we cannot do concurrent execution: while decoding we are ignoring sonar echoes!

2.c.
We could attack this by doing iterleaving "by hand": We split the algorithm for decoding into a number of phases and we add code for attempting to read and deal with sonar echoes in between.

```
decode(*Packet pckt, *Params prms){
 decode1();
 if (SONAR_STATUS & READY == 1){
      dist = SONAR_DATA;
      control(dist, &signal, &params);
      servo_write(signal);
   }
 decode2();
 if (SONAR_STATUS & READY == 1){
      dist = SONAR_DATA;
      control(dist, &signal, &params);
      servo_write(signal);
   }
ETC.
}
```

3.a. Each function call works with its own copy of the arguments, there is no need to protect uses of the arguments with locks.

3.b. Global variables need to be protected as all function calls work on the same memory place and race conditions might occur.

3.c. Each function call works on its own copy of the local variables, there is no need to protect them.

4.

```
typedef struct{
   Object super;
   int value;
   int nrOfReset
}Counter;

#define initCounter() {initObject(),0,0}

int inc(Counter * self, int x){
  self -> value++;
}

int reset(Counter * self, int x){
  self -> value = 0;
  self -> nrOfReset++;
}

int readValue(Counter * self, int x){
  return self -> value;
}

int nrResets(Counter * self, int x){
   return self -> nrOfResets;
}
```

5.

```
typedef struct{
  Object super;
  PIEZO * p;
  int period;
  int duration;
  int running;
} Sound;

#define initSound(p) {initObject(),p,0,0,0}

int start(Sound * self, int x){
   self->running = 1;
   if(self -> duration != 0){AFTER(MSEC(self->duration), self, stop, 0);}
   play(self,0);
}

int stop(Sound * self, int x){
  self -> running = 0;
}

int setDuration(Sound * self, int x){
   self -> duration = x;
}

int setPeriod(Sound *self, int x){
  self->period = x;
}

int play(Sound * self, int x){
  if(self->running){
    ASYNC(self->p, on, 0);
    AFTER(MSEC(self->period), self->p, off, 0);
    AFTER(MSEC(2*self->period), self, play,0);
  }
}
```