FaMAF, Universidad Nacional de Córdoba
Ingeniería de Software I
Segundo Cuatrimestre

Trabajo Práctico N° 6
Testing

1. ¿Cuál es el objetivo de la programación estructurada? Explíquelo.

2. A nivel de diseño detallado, ¿qué métodos de verificación conoce? Descríba brevemente cada uno de ellos.

3. ¿Qué es la refactorización de código? ¿Cuáles son las reglas más importantes para evitar que la refactorización "rompa" el código correcto ya existente?

4. Usted se encuentra en una reunión para la revisión del código: ¿en qué cosas debería enfocar su atención con el fin de tratar de mejorar la calidad del código?

5. ¿Qué determina la clasificación de testing de caja negra o testing de caja blanca?

6. ¿Para qué sirve el grafo causa-efecto en testing de caja negra? Dé un ejemplo de su uso.

7. Dé un pequeño ejemplo de programa erróneo que pase un test por particionado por clases de equivalencias y no pase un test por anáisis de valores límites. Justifique.

8. Dé un pequeño ejemplo de programa erróneo para el cual haya un test que cumpla el el criterio de cobertura de sentencia que detecte el error y un test que cumpla el el criterio de cobertura de ramificación que no lo detecte. Justifique.

9. Considere un video club que tiene dos tipos de socios: abonados y no abonados. Los socios no abonados pueden retirar, de acuerdo a la diponibilidad, cualquier DVD pagando el monto correpondiente a su retiro. Los socios abonados tienen derecho a retirar sin costo (aparte del abono mensual) cualquier DVD disponible siempre que no sea un DVD nuevo, i.e., que haya sido adquirido por el videoclub durante el último mes. En el caso de que un socio abonado desee retirar un DVD nuevo, este debería pagar como un socio no abonado. Construya el grafo de causa y efecto de este problema y la tabla de decisión asociada para determinar los tests necesarios. Tenga en cuenta los escenarios excepcionales (ej.: DVD no disponible, socio moroso, nro. de socio incorrecto, etc.).

10. Considere el siguiente fragmento de código:

```
if((b < a && a > c) || (c > a && a > b))
   m=a;
else if((a > b && b > c) || (c > b && b > a))
   m=b;
else if((a > c && c > b) || (b > c && c > a))
   m=c;
else
   printf(‘‘No middle value.\n’’);
printf(‘‘%d\n’’, m);
```

Construya conjuntos de casos de test para el mismo que cumplan con los criterios de cobertura de ramas y de sentencias, y sabiendo que las variables de entrada son a, b y c, de tipo entero. El programa debería calcular el valor intermedio de las tres variables. ¿Alguno de sus conjuntos de casos de test, permite detectar errores en el programa?

11. Considere el siguiente fragmento de código:

```
int i;
int flag;
flag = 1;
for (i=2; (i<(n/2)) && flag; ) {
   if ((n % i) == 0)
      flag = 0;
   else
      i++;
}

if (flag)
   printf("%d is prime\n", n);
else
```

```
   printf("%d has %d as a factor\n", n, i);
return 0;
```

Construya conjuntos de casos de test para el mismo que cumplan con los criterios de cobertura de sentencias y de ramas, y sabiendo que la variable de entrada es n, de tipo entero.

12. Considere el siguiente fragmento de código:

```
int n,
     lcv,
     flag; /* flag initially is 1 and becomes 0 if we determine that n is not a prime */

for (lcv=2, flag=1; lcv <= (n / 2); lcv++) {
   if ((n % lcv) == 0) {
       if (flag)
         printf("The non-trivial factors of %d are: \n", n);
      flag = 0;
      printf("\t%d\n", lcv);
   }
}
if (flag)
   printf("%d is prime\n", n);
```

Construya conjuntos de casos de test para el mismo que cumplan con los criterios de cobertura de sentencias y de ramas, y sabiendo que la variable de entrada es n, de tipo entero.

13. Considere el siguiente programa:

```
#include <stdio.h>

#define IN_WORD 1 /* currently inside a word */
#define NOTIN_WORD 0 /* currently not in a word */

void main(void)
{
   register int c; /* input char */
   register int nl; /* line count */
   register int nw; /* word count */
   register int nc; /* char count */
   register int state; /* in or not in a word? */

   nl = nw = nc = 0;
   state = NOTIN_WORD;
   while((c = getchar()) != EOF){
      /* got another character */
      nc++;
      /* is it a newline? */
      if (c == '\n')
         nl++;
      /* is it a word separator? */
      if (c == ' ' || c == '\t' || c == '\n')
         /* YES -- change state */
         state = NOTIN_WORD;
      else if (state == NOTIN_WORD){
         /* NO -- we're now in a word; update */
         /* the counter and state if need be */
         state = IN_WORD;
         nw++;
      }
   }

   printf("%6d\t%6d\t%6d\n", nl, nw, nc);
   exit(0);
}
```

Construya conjuntos de casos de test para el mismo que cumplan con los criterios de cobertura de sentencias y de ramas, considerando como entrada la cadena provista por el usuario para analizar.

14. Considere el siguiente fragmento de código:

```
 void selectionSort(int a[], int n)
/* It sorts in non-decreasing order the first N positions of A. It uses
 * the selection sort method.
 */
 {
    int lcv;
    int rh;          /*Elements in interval rh..n-1 are in their final position*/
    int where;    /*Position where we have current maximum*/
    int temp;      /*Used for swapping*/

    for(rh=n-1;rh>0;rh--){
       /*Find position of largest element in range 0..rh*/
       where = 0;
       for (lcv=1;lcv<=rh;lcv++)
          if (a[lcv]>a[where])
             where = lcv;
          temp = a[where];
          a[where] = a[rh];
          a[rh] = temp;
       }
  }
```

Construya conjuntos de casos de test para el mismo que cumplan con los criterios de cobertura de sentencias y de ramas.

15. Construya conjuntos de test para verificar todas las definiciones para los códigos de los Ejercicios 11, 12 y 13.

16. Construya conjuntos de test para verificar todas los usos-c para el código del ejercicio 12.

17. Construya conjuntos de test para verificar todas los usos-p para el código del ejercicio 11.