Aclaraciones sobre el parcial:

1) Comienzo y duración de cada ejercicio
    - **Ejercicios 1 y 2:** comienzan a las 17:05 hs y tienen 60 minutos para hacerlo
    - **Ejercicios 3 y 4:** comienzan a las 18:20 hs y tienen 90 minutos para hacerlo
2) Los ejercicios que no respeten el enunciado serán considerados inválidos
3) Resolver cada ejercicio en el cuadro correspondiente
4) Para aprobar el parcial se debe **sumar 5 (cinco) o más puntos**
5) Ver todos los ejemplos antes de comenzar a resolver el ejercicio
6) Si sienten la necesidad de realizar algún comentario sobre lo que asumieron del enunciado, comentarlo en la respuesta. No se aceptarán aclaraciones por mail o en forma oral
7) Si se detecta de acuerdo a la grabación que están usando otra computadora (Respondus no permite el uso de un segundo monitor) el parcial será anulado
8) Recuerden no presionar el Alt-tab o Command-Tab
9) Cada ejercicio contendrá en la calificación una copia de la respuesta dada y los comentarios sobre la corrección, para que puedan revisarlo
10) Las notas y comentarios sobre cada ejercicio los podrán ver directamente desde el Campus. Se fijará día y horario para revisión. En caso de no entender alguna corrección y no poder concurrir en ese horario, pueden enviar mail solicitando revisión, indicando qué ejercicios necesitan revisar, lo mismo si consideran que hay algo mal corregido (mal corregido significa que consideran que les marcaron como error algo que estaba bien, no que sienten que le bajaron muchos puntos)

En caso de encontrarse con un error de Respondus enviar un mail (de ser posible adjuntando una foto que capture el error en pantalla) a mgarbe@itba.edu.ar y a fmeola@itba.edu.ar indicando el error obtenido.

Ejercicio 1 (1,5 puntos)

Escribir una **macro `NO_ACENTO`** que recibe una expresión y, si esa expresión representa una vocal minúscula acentuada, retorne el ASCII de la vocal sin acento. Si no es una vocal minúscula acentuada retorna el mismo valor.

No usar funciones auxiliares

Ejemplos

```
int c = NO_ACENTO('ó');  // c vale 'o'
c = NO_ACENTO('ó') + 1;  // c vale 'p'
c = NO_ACENTO(1 + 'ú' - 1);  // c vale 'u'
c = NO_ACENTO('r');  // c vale 'r'
c = NO_ACENTO('%');  // c vale '%'
c = NO_ACENTO(64);  // c vale '@', cuyo ASCII es 64
c = NO_ACENTO('Ú');  // c vale 'Ú', porque no es minúscula
```

Ejercicio 2 (3 puntos)

Escribir la función **`depura`** que reciba un vector de `unsigned char` y su dimensión (no es un string *null terminated*). El mismo contiene valores repetidos y no está ordenado. La función debe **dejar en el vector únicamente valores no repetidos y ordenados en forma ascendente**, también debe retornar la nueva dimensión del vector.

Se espera que a esta función se la invoque con vectores de miles de elementos.

Ejemplo de uso (en el ejemplo se colocan pocos valores, pero se espera que en un entorno real de trabajo el vector recibido tenga varios miles de elementos)

```
int
main(void) {
    unsigned char v[] = {1, 3, 1, 5, 2, 1, 5, 1, 0, 100, 95, 100,
3}
    int n = depura(v, sizeof(v) / sizeof(v[0]) );
    assert(n == 7);
    assert(v[0] == 0);
    assert(v[1] == 1);
    assert(v[2] == 2);
    assert(v[3] == 3);
    assert(v[4] == 5);
    assert(v[5] == 95);
    assert(v[6] == 100);

    return 0;
}
```

Ejercicio 3 (2,5 puntos)

Escribir la función **comprime** que recibe un string **s**, un carácter **c** y un vector de enteros **v**. La función debe reemplazar en el string **s** cada secuencia del carácter **c** por un sólo carácter **c**, y dejar en cada posición del vector cuántos caracteres de la secuencia se eliminaron. Además debe retornar la dimensión del vector **v**.

Si el carácter es una letra, tiene que compactar sin distinguir entre mayúsculas y minúsculas, esto es, se considerará que 'A' y 'a' son el mismo carácter

Ejemplo de uso

```
int
main(void) {
   int reps[100];
   char s[] = "AAA aaaabbaa a b aaaa capital federal";
   int dim;
   dim = comprime(s, 'a', reps);
   assert(strcmp(s, "A abba a b a capital federal")==0);
   assert(dim == 8);
   assert(reps[0] == 2); // se eliminan 2 de "AAA"
   assert(reps[1] == 3); // se eliminan 3 de "aaaa"
   assert(reps[2] == 1);
   assert(reps[3] == 0);
   assert(reps[4] == 3);
   assert(reps[5] == 0);
   assert(reps[6] == 0);
   assert(reps[7] == 0);

   dim = comprime(s, 'x', reps);
   assert(strcmp(s, "AAA abba a b a capital federal")==0);
   assert(dim == 0);

   char t[] = "AAAaaaa aAaA";
   dim = comprime(t, 'a', reps);
   assert(strcmp(s, "A a")==0);    // Se deja la primer aparición
   assert(dim == 2);
   assert(reps[0] == 6);
   assert(reps[1] == 3);

   return 0;
}
```

Ejercicio 4 (3 puntos)

Escribir la función **`checkBoard`** que recibe como único parámetro una matriz cuadrada de dimensión `DIM` (es una constante simbólica previamente definida) y retorne 1 cuando la matriz representa un **tablero válido** de un juego al estilo ajedrez o dama.

Un **tablero es válido** cuando cumple **ambas** condiciones mencionadas a continuación:

- **No existen dos posiciones adyacentes (tanto en forma horizontal o vertical) del mismo color**
- **Sólo se utilizaron dos colores en todo el tablero**

Los posibles colores están dados por el enum **`colors`**. Se garantiza que en la matriz estarán únicamente los valores del enum `colors`, no es necesario validarlo.

Ejemplo de uso

```
#define DIM 4

typedef enum colors {black=0, white, blue, red, purple, orange};

int
main(void) {
   unsigned char t[DIM][DIM] = {
           {black, white, black, white},
           {white, black, white, black},
           {black, white, black, white},
           {white, black, white, black}};
   // Es un tablero válido
   assert(checkBoard(t) == 1);

   unsigned char t2[DIM][DIM] = {
           {black, white, black, white},
           {white, black, white, black},
           {black, white, black, white},
           {white, black, white, blue}};
   // No es válido pues se usaron más de dos colores
   assert(checkBoard(t2) == 0);

   unsigned char t3[DIM][DIM] = {
           {black, white, black, white},
           {white, black, white, black},
           {white, black, white, black}};
           {black, white, black, white},
   // No es válido pues hay dos posiciones adyacentes del mismo
color
   assert(checkBoard(t3) == 0);

   return 0;
}
```