SSL 2005.1 – TP Simulación de un Año Académico de SSL – Entrega #3 – TAD Curso

1.1.0.20050531

por José María Sola y Jorge Muchnik.

Este TAD puede ser extendido más adelante para cubrir las necesidades globales del TP.

Requisitos

- Haber aprobado TP #0.
- Haber realizado la Entrega #2.
- Haber leído y comprendido el documento anterior: "**SSL 2005.1 – Presentación General del Trabajo Práctico – Simulación de un Año Académico de SSL**"

Objetivos

- **Especificar** los *valores* y *operaciones* del TAD **Curso** aplicando las herramientas conocidas.
- **Implementar** el TAD **Curso** en una **Biblioteca** aplicando las herramientas conocidas.
- **Probar** la implementación del TAD **Curso** mediante una **aplicación de prueba**.
- Aplicar **archivos binarios** y sus **streams** (**flujos de datos**) asociados de ANSI C.
- Aplicar **campos de bits** de un struct de ANSI C.
- Aplicar los **operadores de bits** en ANSI C para una implementación eficiente.
- **Capturar** las **salidas** de los **procesos de traducción** y de la **aplicación de prueba**.
- **Continuar** con el análisis y el diseño del **TAD Calendario** y del programa de **Simulación**.

Sobre el TAD Curso

Importante: Esta sección es un relato que describe el TAD; **no es su especificació**n.

EL TAD **Curso** representa un conjunto de **alumnos** que cursa SSL durante un **cuatrimestre de un año** determinado y con un **Docente** determinado. Cursar implica **asistir** y **rendir evaluaciones**.

Un curso se identifica por un **código de curso** de 5 caracteres (K2001, K2102, etc.). Los alumnos se identifican y representan por su **legajo**, un número entero perteneciente al intervalo abierto (40.000, 200.000). Las calificaciones para las evaluaciones son **Aprobado** ó **Desaprobado**. El Docente se representa por sus nombres y apellidos, de longitud variable.

Para que un alumno pueda **regularizar** su cursada, deberá asistir como **mínimo al 75% de las clases** y **aprobar** los tres tipos de evaluación en cualquiera de las **tres oportunidades** (evaluación original, 1er recuperatorio y 2do recuperatorio). La simplificación del simulación establece que la **fecha de regularización** es la misma para todos los alumnos que hayan regularizado.

Existen tres tipos de evaluación y dos recuperatorios para cada una. Los tipos de evaluación son **Evaluación Grupal Escrita**, **Evaluación Grupal Oral** y **Evaluación Individual**. Los primeros recuperatorios para las tres evaluaciones se toman en una misma fecha, los segundos recuperatorios para las tres evaluaciones se toman, también en conjunto, en otra fecha. Todos los alumnos deben aprobar los tres tipos de evaluación.

¿Cómo se definen los alumnos inscriptos? Mediante un conjunto de legajos, con cardinalidad igual a la cantidad de inscriptos.

¿Cómo se define la asistencia al curso por parte de los alumnos? Mediante un conjunto de pares ordenados (legajo, asistencia); la asistencia de cada alumno es una secuencia de hasta 32 valores lógicos que representen "Presente" ó "Ausente".

¿Cómo se definen las calificaciones de los alumnos en las diferentes evaluaciones? Para cada tipo evaluación habrá un conjunto de legajos pares ordenados (legajo, calificación); la calificación es un valor lógico que representa "Aprobado" ó "Desaprobado".

Como resultado del análisis de las frases anteriores, la n-upla que representa un valor del TAD Curso queda definida de esta manera: **(Código de curso, Año, Cuatrimestre, Docente, Cantidad de inscriptos, Alumnos, Asistencias, Evaluaciones grupales escritas, Evaluaciones grupales orales, Evaluaciones individuales, Primeros recuperatorios, Segundos recuperatorios, Fecha de regularización)**.

A nivel de la implementación, un único arreglo de tamaño definido dinámicamente será utilizado para representar todos los conjuntos de la n-upla. Los elementos del arreglo serán estructuras compuestas por el legajo, la asistencia y las evaluaciones de cada alumno. La asistencia por parte de un alumno a las 32 clases del cuatrimestre se representarán en 4 bytes (32 bits). Para manipular esta asistencia se construirán funciones que **ocultarán** el manejo a bajo nivel de los 32 bits.

Operaciones

La especificación de cada operación debe incluir todas las precondiciones y poscondiciones. Realizar un análisis correcto y exhaustivo de estas condiciones para una especificación correcta y sin ambigüedades de las operaciones.

Creación

- **Creación**. Dados un código de curso, un año (Natural), un cuatrimestre (1 ó 2), los nombres y apellidos del docente (cadena de longitud variable) y la cantidad de inscriptos, produce un valor del TAD Curso, con la fecha de regularización indefinida. Utilizar reserva dinámica de memoria en la implementación de los nombres y apellidos del docente y para el conjunto de alumnos.

Consulta

- **Regularización** (predicado). Dados un legajo y un valor del TAD Curso, retorna si el alumno regularizó la asignatura o no. Una precondición de esta operación es la aplicación de la operación de Cierre. La implementación más eficiente se logra consultado el campo bit "regularizo". *Contrastar con la operación privada Consulta de la Regularización de un Alumno (IsRegular) en la sección Operaciones Privadas.*
- **Fecha de regularización**. Dados un legajo y un valor del TAD Curso, retorna el número de día y el número de mes de regularización. *No es precondición* que el alumno haya regularizado, ya que si no regularizó, retorna 0 tanto para el día como para el mes. Conceptualmente, dos alumnos pueden haber regularizado en diferente fecha, pero como hipótesis de simplificación de la simulación, la fecha de regularización es común.

Ambas operaciones deben utilizar la operación privada *GetAlumno* que implementará la búsqueda de alumno. Ver sección *Operaciones Privadas*.

Acceso

- **Código**. Dado un valor del TAD Curso, retorna su código.
- **Año**. Dado un valor del TAD Curso, retorna su año.
- **Cuatrimestre**. Dado un valor del TAD Curso, retorna su cuatrimestre.
- **Docente**. Dado un valor del TAD Curso retorna una cadena con el nombre y apellido.
- **Cantidad de inscriptos**. Dado un valor del TAD Curso, retorna la cantidad de inscriptos.

Modificación

- **Inscripción de un alumno**. Dados un legajo y un valor del TAD Curso, retorna un Curso idéntico al dato pero con el conjunto de alumnos inscriptos diferente.
- **Tomar Asistencia**. Dados un valor del TAD Curso y un número de clase, retorna un nuevo Curso con la asistencia para esa clase. En los días con evaluación también se toma asistencia. En la implementación se deben utilizar las operaciones privadas *GenerarAsistencia* y *SetAsistencia*.
- **Evaluaciones**. Dado un valor del TAD Curso, retorna un nuevo Curso con calificaciones para la evaluación correspondiente. Si un alumno está ausente, su calificación es desaprobado. Analizar correctamente precondiciones y poscondiciones. En la implementación se debe utilizar las operación privada *GenerarAprobacion*. Las operaciones de este grupo son:
  1. *Evaluación Grupal Escrita*
  2. *Evaluación Grupal Oral*
  3. *Evaluación Individual*
  4. *Primer Recuperatorio*
  5. *Segundo Recuperatorio*
- **Cierre.** Dados un valor del TAD Curso, un día (natural) y un mes (natural), retorna un nuevo Curso con la fecha de regularización definida, con los alumnos que regularizaron la asignatura y con los alumnos que no. Analizar correctamente precondiciones y poscondiciones. En la implementación se debe utilizar la operación privada *Consulta de la Regularización de un Alumno (IsRegular).*

Salidas por streams de texto y binarios

Estas operaciones reciben como dato un Curso y un stream, de texto o binario, abierto para escritura. Las operaciones escriben en sus streams dato, y retornan el mismo stream pero avanzado según los elementos que se hayan escrito. Para la aplicación de prueba, el stream dato de las operaciones que escriben en flujos de texto será `stdout`.

Asimismo, para la aplicación de prueba se construirá una función extra que permitirá evaluar el correcto funcionamiento de la operación que escribe en un flujo binario. Esta operación leerá un valor del TAD Curso, previamente almacenado en un archivo binario, con la operación *Acta Binario*.

- **Emisión de Acta**. Es el listado que se produce al terminar un curso. Esta información académica se genera en dos formatos. 1. Texto: Para su lectura y 2. Binario: Para su almacenamiento e intercambio eficiente.
  1. ***Flujo de Texto.***
  Escribe un encabezado con datos del curso, un cuerpo con la asistencia y las evaluaciones de los alumnos, y un pie con los totales.
  Encabezado. 4 líneas, incluyendo dos con los nombres de las columnas
  ```
  "Asignatura: SSL - Curso: %s – Año %d Cuatrimestre %s\n"
  "Docente: %s\t
  "Legajo\tAsistencia\t\t\t\tEvaluaciones\tRegularización\n"
  "\t1234567890 1234567890 1234567890 12\tE O I 1 2\tFecha\n"
  ```
  Cuerpo. Una línea por cada alumno, con el legajo, vector de asistencia, los 3 tipos de evaluaciones, los dos recuperatorios y la fecha de regularización. La presencia se marca con los caracteres 'P' y 'A', para *Presente* y *Ausente* respectivamente. La calificación con 'A' y 'D' para *Aprobado* y *Desaprobado* respectivamente. Sino rindió algún recuperatorio se completa con '-'. El mes y el día tienen un ancho de dos dígitos, se completan con ceros cuando el día y/o el mes es de un solo dígito. Sino regularizó, en lugar de la fecha se escribe "RECURSA".
  Estructura de formato de los dos tipos de línea:
  ```
  "%s\t%c%c%c%c%c … %c%c %c%c\t%c %c %c %c %c \t%02d/%02d"
  "%s\t%c%c%c%c%c … %c%c %c%c\t%c %c %c %c %c \tRECURSA"
  ```
  Ejemplos:
  ```
  "110811\tPPPPAPPPAP PAAPPPPPAP PPPPAPAPPP PP\tA A D D A\t14/07\n"
  "110824\tPPPAPAPPPP PPPPAPPPPP APPPPPPPPP PP\tA A A - -\t14/07\n"
  "110828\tPPPAPAAAAA AAAAAAAAAA AAAAAAAAAA AA\tD D D - -\tRECURSA\n"
  "110829\tPPPAPAPPPP PPPPAPPPPP APPPPPPPPP PP\tA A A - -\t14/07\n"
  "110853\tPPPPPPAPPP PAPAPPPPPP PPPPPPPPPA PA\tD D A A -\t14/07\n"
  "110857\tPPPPPAPPPP PPPPAPPPPP PPPAPPPPPP PP\tD D D - -\tRECURSA\n"
  ```
  Pie. 1 línea.
  ```
  "Inscriptos: %d\tRegularizados: %d\tRecursantes: %d\n"
  ```

  2. ***Flujo Binario.***
  Escribe una secuencia de valores que representa todos los datos de un curso luego de su cierre. La secuencia es la siguiente: código de curso, año-cuatrimestre, longitud nombre docente, nombre docente, cantidad de inscriptos, $alumno_1$, $alumno_2$, $alumno_n$.
  La secuencia tiene partes de longitud fija y otras de longitud variable. El código es de longitud fija, mientras que el nombre del docente no. La longitud del nombre del docente está dada por el 3er campo. El 5to campo define la longitud de la siguiente sección también variable, que es los datos de cada alumno.
  El campo año-cuatrimestre es de la forma año*10 + cuatrimestre. Ejemplos: 20042; 20051; 20052.
  El campo $alumno_n$ es a su vez una secuencia formada por:
  legajo, 4 bytes para la asistencia, 5 bytes para las evaluaciones, y la fecha de regularización. La fecha se almacena como mes*100 + día, por ejemplo 630 ó 1207.
  Las cadenas se almacenan sin su centinela terminador.
- **Emisión de Regularizados**. Una línea con el legajo y fecha de regularización de cada alumno que haya regularizado la asignatura.
- **Emisión de Recursantes**. Una línea con el legajo de cada alumno que no haya regularizado la asignatura.

Privadas de la Implementación

Las siguientes operaciones no forman parte de la especificación del TAD. Deben ser declaradas con el especificador `static` para encapsularlas dentro de la biblioteca. Son funciones necesarias para una **mejor estructuración y diseño de la implementación**.

En la implementación del TAD, en el archivo fuente que define la biblioteca y por encima del prototipo de cada una estas funciones, debe colocarse un **comentario con la documentación de la función** indicando, entre otras cosas: propósito de la función, semántica de los parámetros *in*, *out* e *inout*.

Manipulación del Arreglo de Alumnos

Estas funciones permiten una mayor abstracción al acceder a la asistencia y evaluaciones de cada alumno del arreglo, pero no forman parte de la sección pública de la biblioteca.

Búsqueda y Obtención de un Alumno (*GetAlumno*)

Dado un valor del TAD Curso y un legajo, esta función retorna un alumno. Para lograr un implementación simple y eficiente la operación *Inscribir Alumno* (i.e. inserción rápida, al final), y para simplificar la construcción de esta función, la búsqueda será lineal (utilizar `strcmp`). La función retorna un puntero a un alumno, y de no encontrarlo, retorna `NULL`.

Consulta de la Regularización de un Alumno (*IsRegular*)

Dado un alumno y la cantidad real de clases, esta función retorna un valor lógico indicando si el alumno cumple con lo requerido para regularizar la materia (asistencia y evaluaciones). Esta operación deberá ser utilizada en la implementación de la operación *Cierre* y debe utilizar la operación privada *Consulta de la Cantidad de Clases que un Alumno estuvo Presente.*

Consulta de la Cantidad de Clases que un Alumno estuvo Presente

Dado un alumno, esta función retorna un natural indicando la cantidad de clases a la que el alumno concurrió durante el cuatrimestre. Esta función debe utilizarse cuando se establece la regularización o no de un alumno en la operación privada *Regularización de un alumno (IsRegular)* y debe utilizar la operación privada *Asistencia a una Clase Determinada.*

Asistencia a una Clase Determinada

Estas tres funciones serán invocadas por otras funciones de la implementación. Reciben un alumno y un número de clase (de 1 a 32).

- **Modificación**. *Set* y *Reset*.
- **Consulta**. *Get.*

Las funciones de modificación *setean* ó *resetean* el valor lógico que representan la asistencia de determinada clase. Ambas se invocan al tomar asistencia. *Ver operación pública Tomar Asistencia.*

La función de consulta obtiene si un alumno asistió o no a una clase determinada, retorna la un valor lógico que lo indica. Se invoca al identificar la cantidad de clases asistidas. *Ver operación privada Consulta de la Cantidad de Clases que un Alumno estuvo Presente.*

Las tres funciones se valen de los operadores de bits (e.g. << y >>) que provee ANSI C.

Estadísticas

Ambas operaciones utilizan el tipo de dato `int` para representar *valores lógicos*. Serán invocadas por las implementaciones de la *operación de asistencia* y las *5 operaciones de evaluación.*

Las estadísticas indican que el 75% de los alumnos asisten al 85% de las clases mientras que el 25% de los alumnos restantes asisten al 55% de las clases.

La estadísticas también indican que el 55% de los alumnos aprueban una evaluación, cualquiera fuera el tipo de evaluación (individual, grupal escrita, grupal oral, primer recuperatorio o segundo recuperatorio).

- `static int` **`GenerarAsistencia`**`(void)`.
- `static int` **`GenerarAprobacion`**`(void)`.

Requeridas para esta la Implementación

- **Destrucción**. Dada una variable del TAD Curso, libera los recursos previamente asignados.

Restricciones y Guías para la Implementación

La asistencia de cada alumno a las 32 clases se representa con 4 bytes (32 bits). Las evaluaciones de cada alumno se representan con bit-fields (campos de bits). La asignación y el acceso se realiza como cualquier otro campo de la estructura.

```
typedef struct {
   const char    legajo[6+1];
   unsigned char asistencia[4]; /* 32 bits */
   unsigned int  evaluacionGrupalEscrita : 1; /* 1 bit */
   unsigned int  evaluacionGrupalOral    : 1; /* 1 bit */
   unsigned int  evaluacionIndividual    : 1; /* 1 bit */
   unsigned int  primerRecuperatorio     : 1; /* 1 bit */
   unsigned int  segundoRecuperatorio    : 1; /* 1 bit */
   unsigned int  regularizo              : 1; /* 1 bit */
} Alumno;

typedef struct {
    const char   codigo[5+1];
    int          anioCuatrimestre; /* anio*10 + cuatrimestre */
    const char*  docente;
    int          cantidadRealDeClases;
    int          inscriptos;
    Alumno*      alumnos; /* puntero al primer elemento de un arreglo de alumnos */
   int           fechaDeRegularizacion; /* mes * 100 + dia */
} Curso;
```

Identificadores para las Funciones que Implementan las Operaciones y guías para algunos Prototipos

```
1. Curso Curso_Crear(
     const char* unCodigo,
     int         unAnio,
     int         unCuatrimestre,
     const char* unProfesor,
     int         unaCantidadDeInscriptos
   );

2. int Curso_IsRegular(
     const Curso* unCurso, /* in */
     const char*  unLegajo /* in */
   );

3. void Curso_GetFechaDeRegularizacion(
     const Curso* unCurso,  /* in */
     const char*  unLegajo, /* in */
     int*         unDia,    /* out */
     int*         unMes,    /* out */
   );

4. int Curso_InscribirAlumno(
     const Curso* unCurso, /* inout */
     const char*  unLegajo /* in */
   );

5. void Curso_TomarAsistencia(
     Curso* unCurso.        /* inout */
     int    unNumeroDeClase /* in */
   );

6. void Curso_TomarEvaluacionGrupalEscrita(
     Curso* unCurso /* inout */
   );

7. Curso_TomarEvaluacionGrupalOral

8. Curso_TomarEvaluacionIndividual

9. Curso_TomarPrimerRecuperatorio

10. Curso_TomarSegundoRecuperatorio

11. void Curso_Cerrar(
      Curso* unCurso, /* inout */
      int    unDia,   /* int */
      int    unMes    /* int */
    );

12. void Curso_EmitirActaTexto(
      const Curso* unCurso, /* in */
      FILE*        unStream /* inout */
     );
```

```
13. Curso_EmitirActaBinario

14. Curso_EmitirRegularizadosTexto

15. Curso_EmitirRecursantesTexto

16. const char* Curso_GetCodigo(
      const Curso* unCurso /* in */
    );

17. Curso_GetAnio

18. Curso_GetCuatrimestre

19. Curso_GetDocente

20. Curso_GetCantidadDeInscriptos

21. void Curso_Destruir(
      const Curso* unCurso /* in */
    );

/* Operaciones privadas */

22. static void SetAsistencia(
      Alumno* unAlumno, /* inout */
      int     unaClase  /* in */
    );

23. static void ResetAsistencia

24. static int GetAsistencia(
      const Alumno* unAlumno, /* in */
      int           unaClase  /* in */
    );

25. static int GetCantidadDeClasesPresente(
      const Alumno* unAlumno /* in */
    );

26. static int IsRegular(
      const Alumno* unAlumno,            /* in */
      int           cantidadRealDeClases /* in */
    );

27. static Alumno* GetAlumno( /* Búsqueda y obtención */
      const char*  unLegajo, /* in */
      const Curso* unCurso   /* in */
    );

28. static int GenerarAsistencia(void);

29. static int GenerarAprobacion(void);
```

La implementación del TAD Curso busca ser eficiente con respecto al espacio en memoria principal. Por eso, trata de usar bits para guardar valores lógicos, en lugar de utilizar otros tipos de datos.

Más precisamente, la asistencia a las 32 clases puede representarse con, por ejemplo: 32 chars (32 bytes) ó con 32 ints (`sizeof(int) * 32`, 128 bytes en BCC32). Pero en esta implementación se decidió utilizar 4 chars, esto es, sólo 4 bytes o 32 bits. Teniendo en cuenta que esa cantidad es por alumno y que un curso puede contener varios alumnos y varios cursos de varios años pueden ser manejados por un sistema mayor más complejo, se evita el derroche de memoria y se logra un eficiente uso de la misma.

El lenguaje C provee un conjunto de Operadores para el manejo de Bits. Su uso es simple pero no es común en todos los ámbitos, por eso se incluimos referencias y una guía para la implementación.

Los Campos de Bits tienen muchos puntos en común con campos de otro tipo (su uso es igual, solo varía su declaración) por lo que su estudio y aplicación queda del lado de ustedes.

Campos de Bits

- [M3] 1.5.2. Declaraciones - <decla "struct">
- [K&R] 6.9. Bit-fields

Operadores de Bits

- [M3] 1.5.1. Operaciones - Precedencia y Asociatividad - operadores de desplazamiento, AND, OR y XOR de bits
- [K&R] 2.9. Bitwise Operators

Ejemplo:

```
/* Corre 2 posiciones hacia la derecha los bits del
valor 4, completando con ceros.*/
unsigned char c = 4 >> 2;
```

Debajo se presentan implementaciones guía (ustedes deben chequearlas) de las operaciones privada de asistencia, que hacen uso de operadores de bits. Cabe notar que cada una de las tres operaciones se implementa con sólo 3 sentencias.

Estudien, comprendan, validen y, en todo caso, corrijan esta implementación guía.

```c
/* AsistenciaEnBits
 * Implementa y prueba las operaciones privadas
 * para el manejo del arreglo de 32 bits que
 * representa la asistencia a las 32 clases.
 * JMS 20050530
 */
#include <stdio.h>  /* printf */
#include <stdlib.h> /* EXIT_SUCCESS */

typedef struct {
 const    char legajo[6+1];
 unsigned char asistencia[4]; /* 32 bits */
 unsigned int  evaluacionGrupalEscrita : 1; /* 1 bit */
 unsigned int  evaluacionGrupalOral    : 1; /* 1 bit */
 unsigned int  evaluacionIndividual    : 1; /* 1 bit */
 unsigned int  primerRecuperatorio     : 1; /* 1 bit */
 unsigned int  segundoRecuperatorio    : 1; /* 1 bit */
 unsigned int  reguralizo              : 1; /* 1 bit */
} Alumno;

/* Operaciones de asistencia
   que se utilizarán en el TAD Curso */
/* Dado un alumno y una clase, consulta si el alumno
   estuvo presente */
static int GetAsistencia(
 const Alumno* unAlumno, /* in */
 int           unaClase  /* in */
);
/* Dado un alumno, establece en 1 (presente) el bit
   correspondiene a la clase dada */
static void SetAsistencia(
 Alumno* unAlumno, /* inout */
 int     unaClase  /* in */
);
/* Dado un alumno, reestablece en 0 (ausente) el bit
   correspondiene a la clase dada */
static void ResetAsistencia(
 Alumno* unAlumno, /* inout */
 int     unaClase  /* in */
);

/* Función complementaria para la prueba */
void MostrarAsistencia(const Alumno* unAlumno /* in */);

/* Prueba de las operaciones de asistencia */
int main(void){
 Alumno unAlumno;
 int i;

 /* Marca "manualmente" a unAlumno como
    presente en las clases:
    1, 9, 10, 14, 17, 24, 25, 26,
    27, 28, 29, 30, 31 y 32 */
 unAlumno.asistencia[0] =   1; /* 0x01 0000 0001 */
 unAlumno.asistencia[1] =  35; /* 0x23 0010 0011 */
 unAlumno.asistencia[2] = 129; /* 0x81 1000 0001 */
 unAlumno.asistencia[3] = 255; /* 0xFF 1111 1111 */
 MostrarAsistencia(&unAlumno);

 /* Marca a todas las clases como presente */
 for(i = 1; i <= 32; i++)
  SetAsistencia(&unAlumno, i);
 MostrarAsistencia(&unAlumno);

 /* Marca a todas las clases como ausente */
 for(i = 1; i <= 32; i++)
  ResetAsistencia(&unAlumno, i);
 MostrarAsistencia(&unAlumno);

 /* Marca que unAlumno asistió a la clase 19 */
 SetAsistencia(&unAlumno, 19);
 MostrarAsistencia(&unAlumno);

 /* Marca que unAlumno no asistió a la clase 19 */
 ResetAsistencia(&unAlumno, 19);
 MostrarAsistencia(&unAlumno);

 return EXIT_SUCCESS;
}

static int GetAsistencia(
 const Alumno* unAlumno, /* in */
 int           unaClase  /* in */
){
 /* obtiene uno de los 4 grupos que contienen
    8 clases representados por 8 bits */
 char asistencia =
   unAlumno->asistencia[ (unaClase-1) / 8 ];

 /* Corre el bit de la clase buscada todo
    a la izquierda, agregando ceros a la derecha */
 asistencia <<= 7 - (unaClase-1) % 8;

 /* Corre el bit de la clase buscada todo a la derecha,
    agregando ceros a la izquierda */
 return asistencia >>= 7; /* 0 ó 1 */
}

static void SetAsistencia(
 Alumno* unAlumno, /* inout */
 int     unaClase  /* in */
){
 /* Establece una máscara: todos los bits en cero
    menos el de la clase buscada */
 char mascara = (char)( 1 << ( (unaClase-1) % 8 ) );

 /* obtiene uno de los 4 grupos que contienen
    8 clases representados por 8 bits y le aplica
    la máscara con el operador de bit OR */
 unAlumno->asistencia[ (unaClase-1) / 8 ] |= mascara;

 return;
}

static void ResetAsistencia(
 Alumno* unAlumno, /* inout */
 int     unaClase  /* in */
){
 /* Establece una máscara: todos los bits en uno
    menos el de la clase buscada */
 char mascara =
    (char)( ~( 1 << ( (unaClase-1) % 8 ) ) );

 /* obtiene uno de los 4 grupos que contienen
    8 clases representados por 8 bits y le aplica
    la máscara con el operador de bit AND */
 unAlumno->asistencia[ (unaClase-1) / 8 ] &= mascara;

 return;
}

/* Muestra las 32 clases, 8 por línea, 4 líneas
   e indica con una 'X' las clases a las
   que unAlumno asistió */
void MostrarAsistencia(const Alumno* unAlumno /* in */){
 int i;

 for(i = 1; i <= 32; i++)
  printf(
   "%02d %c%c",
   i,
   GetAsistencia(unAlumno, i) ? 'X' : ' ',
   i % 8 ? '\t' : '\n'
  );

 printf("\n\n"); /* separador */

 return;
}

/* Salida
01 X  02    03    04    05    06    07    08
09 X  10 X  11    12    13    14 X  15    16
17 X  18    19    20    21    22    23    24 X
25 X  26 X  27 X  28 X  29 X  30 X  31 X  32 X


01 X  02 X  03 X  04 X  05 X  06 X  07 X  08 X
09 X  10 X  11 X  12 X  13 X  14 X  15 X  16 X
17 X  18 X  19 X  20 X  21 X  22 X  23 X  24 X
25 X  26 X  27 X  28 X  29 X  30 X  31 X  32 X


01    02    03    04    05    06    07    08
09    10    11    12    13    14    15    16
17    18    19    20    21    22    23    24
25    26    27    28    29    30    31    32


01    02    03    04    05    06    07    08
09    10    11    12    13    14    15    16
17    18    19 X  20    21    22    23    24
25    26    27    28    29    30    31    32


01    02    03    04    05    06    07    08
09    10    11    12    13    14    15    16
17    18    19    20    21    22    23    24
25    26    27    28    29    30    31    32
*/
```

Conceptos de ANSI C necesarios

- Los necesarios para la entrega anterior.
- Archivos binarios.
- Entrada/Salida Standard con streams binarios de ANSI C.
- Campos de Bits.
- Operadores de Bits de ANSI C.

Sobre la Prueba

Se debe diseñar una aplicación de prueba con datos constantes, que verifique la corrección de todos los casos posibles. Utilice el concepto de partición de conjuntos, y pruebe los casos extremos y un caso representativo de cada partición.

Utilice **expresiones que evalúen las poscondiciones** para comprobar la correcta implementación de las operaciones.

Para las pruebas de las operaciones de streams de texto, utilizar `stdout` como argumento.

Se construirá la función
`Curso ReadCurso(FILE* unStream)`
que retorna un valor del TAD Curso idéntico al Curso que se encuentra listo para leer en el flujo de entrada binario apuntado por unStream. Esta función no forma parte de TAD Curso. *Ver la anterior sección "Salidas por streams de texto y binarios"*.

Otras Restricciones

- Para la implementación debe utilizarse el lenguaje y los elementos de la biblioteca estándar especificados en ANSI C [M3].
- Luego de que la biblioteca y la aplicación de prueba hayan sido construidas con la herramienta de desarrollo elegida por el equipo, deben ser verificadas mediante la herramienta de desarrollo “Borland C++ Compiler 5.5 with Command Line Tools” (BCC32) configurada tal como dicta la cátedra [TP #0].
- Los procesos de compilación con BCC32 no deben emitir warnings (mensajes de advertencia) ni, por supuesto, mensajes de error.
- Los identificadores de las funciones que implementan las operaciones, de las variables, tipos y demás elementos así como los nombres de los archivos deben ser los indicados a lo largo de este enunciado.
- Cualquier decisión que el equipo tome sobre algún punto no aclarado en el enunciado debe ser agregada como hipótesis de trabajo.

Sobre la Entrega

Tiempo

- Siete (7) días después de analizado el enunciado en clase.

Forma

El trabajo debe presentarse en hojas **A4 abrochadas** en la esquina superior izquierda. En el encabezado de cada hoja debe figurar el **título** del trabajo, el título de **entrega**, el código de **curso** y los **apellidos** de los integrantes del equipo. Las hojas deben estar enumeradas en el pie de las mismas con el formato “**Hoja n de m**”.

El código fuente de cada componente del TP debe comenzar con un comentario encabezado, con todos los datos del equipo de trabajo: **curso**; **legajo**, **apellido** y **nombre** de cada integrante del equipo y **fecha de última modificación**. La fuente a utilizar en la impresión debe ser una fuente de ancho fijo (e.g. `Courier new`, `Lucida Console`).

1. TAD Curso
1.1. **Especificación**. Especificación completa, extensa y sin ambigüedades de los valores y de las operaciones del TAD.
1.2. **Implementación**
1.2.2. Biblioteca que implementa el TAD
1.2.2.1. Listado de código fuente del archivo encabezado, **parte pública**, **Curso.h.**
1.2.2.2. Listado de código fuente del la definición de la Biblioteca, **parte privada**, **Curso.c.** Las funciones privadas deberán ser precedidas por su documentación, un comentario con la documentación de la función indicando, entre otras cosas, propósito de la función, semántica de los parámetros *in*, *out* e *inout*.
1.2.3. **Salidas**. Captura impresa de la salida del proceso de traducción (BCC32 y TLIB). Utilizar fuente de ancho fijo.

2. Aplicación de Prueba
2.1. **Código Fuente**. Listado del código fuente de la aplicación de prueba, **CursoAplicacion.c.**
2.2. **Salidas**
2.2.1. Captura impresa de la salida del proceso de traducción (BCC32). Utilizar fuente de ancho fijo.
2.2.2. Captura impresa de las salidas de la aplicación de prueba. Utilizar fuente de ancho fijo.

3. Copia Digitalizada
CD ó disquette con copia de solamente los 3 archivos de código fuente (Curso.h, Curso.c y CursoAplicacion.c).
no se debe entregar ningún otro archivo.

4. Formulario de Seguimiento de Equipo.