OBJETIVOS

- Implementar en lenguaje C algunos de los tipos del trabajo práctico.
- Transformar algunos de los métodos Java que se han realizado en funciones en lenguaje C.
- Crear proyectos a partir de makefiles.

ESTRUCTURA DEL PROYECTO C

Vamos a implementar en lenguaje C una parte del proyecto desarrollado en Java. Para ello debemos crear en primer lugar un proyecto C. Con objeto de poder generar distintos ejecutables (uno por cada tipo a testear), vamos a crear el proyecto de tipo *MakeFile Project*.

Este tipo de proyecto permite usar la herramienta *make* (o, en nuestro caso, mingw32-make) que sirve para gestionar las dependencias entre ficheros y dirigir la compilación y generación de ejecutables automáticamente. Para crear el proyecto con Eclipse seleccione en el menú *File New > C Project*. Luego escoja *MakeFile Project>Empty Project* en el recuadro *Project Type*, y *MinGW GCC* en el recuadro *ToolChains* (Figura 1). Nombre su proyecto de la forma **FP1415-C-*uvus***, donde *uvus* es su usuario virtual de la Universidad de Sevilla (por ejemplo, `FP1415-C-demouser`)

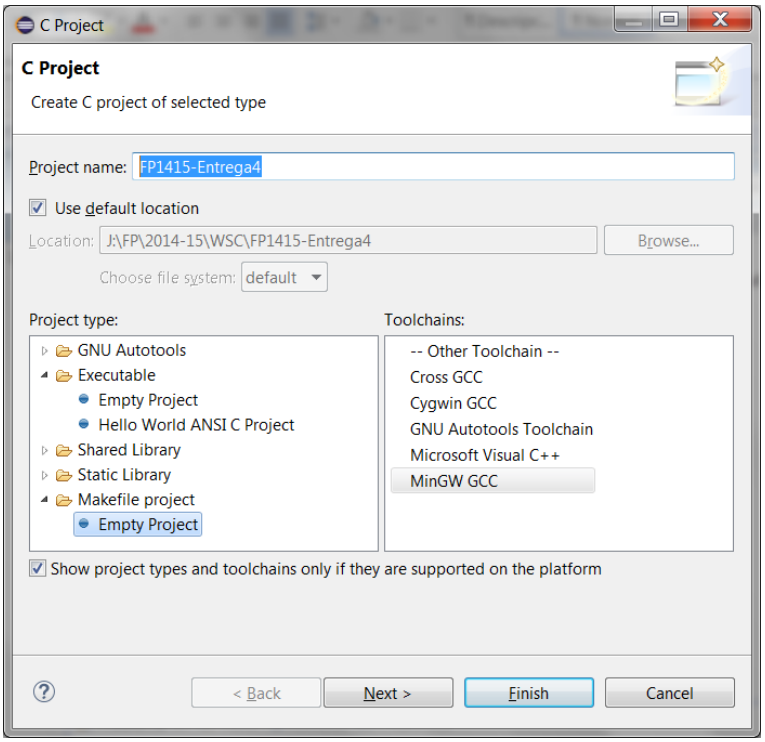

Figura 1. Configuración de proyecto Makefile

Una vez creado el proyecto, hay que cambiar sus propiedades para que utilice la herramienta *make* proporcionada con *MinGW* (`mingw32-make`). Para ello pulse con el botón derecho del ratón sobre el nombre del proyecto, y seleccione la opción *Properties*. Le aparecerá la ventana de un asistente.

En la parte izquierda de la ventana seleccione *C/C++ Build*, y en la parte derecha seleccione la pestaña *Builder Settings*. En la sección *Builder* desmarque la casilla *Use default build command* y en

el recuadro de texto *Build command* escriba `mingw32-make`, tal y como se muestra en la Figura 2. Pulse *Apply* y después *Ok*.

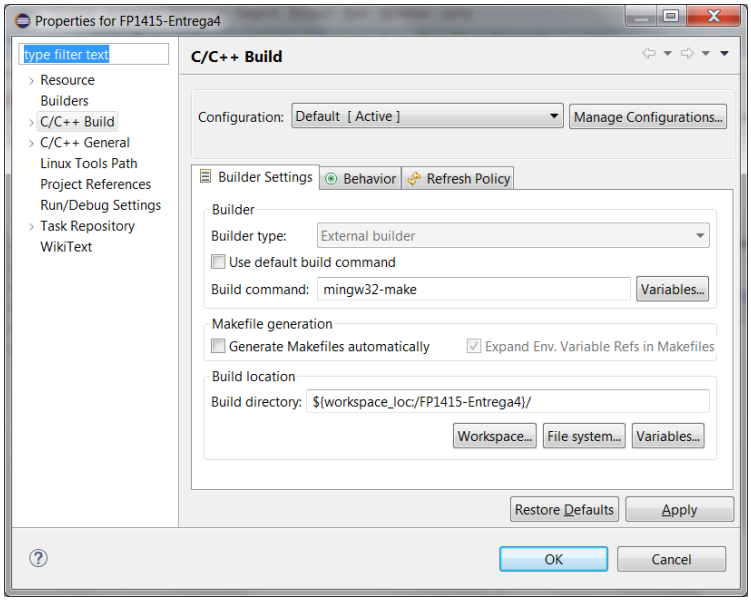

Figura 2. Configuración de propiedades del proyecto para usar mingw32-make

Copie en el directorio raíz de su proyecto el archivo `makefile` que puede descargar de la Enseñanza Virtual. Además, para estructurar mejor el proyecto, cree una carpeta llamada `includes`, en la que deberá almacenar todos los archivos de cabecera (*.h) creados en su proyecto (Figura 3). Cree también una carpeta `res` para almacenar los ficheros de texto de asignaturas y espacios.

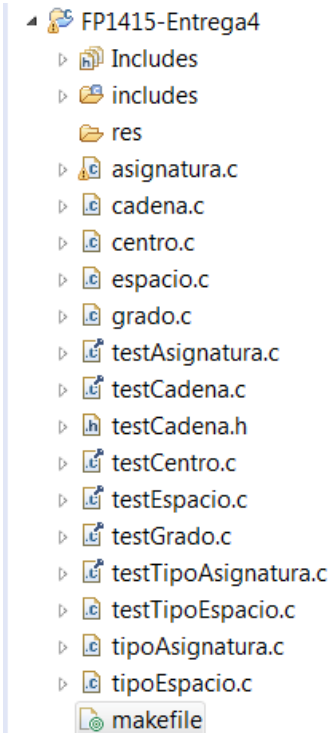

Figura 3. Archivos y carpetas del proyecto

El proyecto contendrá los ficheros que se relacionan a continuación, con el contenido que se indica en cada uno de ellos:

Tipos Cadena y Logico:

- **`cadena.h`**: declaración del tipo Cadena. El máximo número de caracteres de las cadenas será 256, aunque este valor puede cambiar en el futuro. Se incluirá también el siguiente prototipo de función:

```
void quitaSaltoDeLinea(Cadena c);
```

- **`cadena.c`**: definición de la función `quitaSaltoDeLinea`. Esta función debe eliminar el último carácter de la cadena recibida, siempre y cuando dicho carácter sea un salto de línea.
- **`logico.h`**: declaración del tipo enumerado `Logico`, con sus dos valores posibles FALSO y CIERTO.

Tipo Asignatura:

- **`tipoAsignatura.h`**: declaraciones de tipos y funciones relacionadas con el tipo TipoAsignatura.
- **`tipoAsignatura.c`**: definición de las funciones declaradas en `tipoAsignatura.h`.
- **`asignatura.h`**: declaraciones de constantes, tipos y funciones relacionadas con el tipo Asignatura.
- **`asignatura.c`**: definición de las funciones declaradas en `asignatura.h`.

Tipo Grado:

- **`grado.h`**: declaraciones de constantes, tipos y funciones relacionadas con el tipo Grado.
- **`grado.c`**: definición de las funciones declaradas en `grado.h`.

Tipo Espacio:

- **`tipoEspacio.h`**: declaraciones de tipos y funciones relacionadas con el tipo TipoEspacio.
- **`tipoEspacio.c`**: definición de las funciones declaradas en `tipoEspacio.h`.
- **`espacio.h`**: declaraciones de constantes, tipos y funciones relacionadas con el tipo Espacio.
- **`espacio.c`**: definición de las funciones declaradas en `espacio.h`.

Tipo Centro:

- **`centro.h`**: declaraciones de constantes, tipos y funciones relacionadas con el tipo Centro.
- **`centro.c`**: definición de las funciones declaradas en `centro.h`.

Ficheros de test:

- **`testCadena.c`**: pruebas del tipo Cadena.
- **`testTipoAsignatura.c`**: pruebas del tipo TipoAsignatura.
- **`testAsignatura.c`**: pruebas del tipo Asignatura.
- **`testGrado.c`**: pruebas del tipo Grado.
- **`testTipoEspacio.c`**: pruebas del tipo TipoEspacio.
- **`testEspacio.c`**: pruebas del tipo Espacio.
- **`testCentro.c`**: pruebas del tipo Centro.

Veamos con detalle cada uno de los tipos que tenemos que definir.

EL TIPO TIPOASIGNATURA

Se desea disponer de un tipo enumerado `TipoAsignatura`, con los valores `ANUAL`, `PRIMER_CUATRIMESTRE` y `SEGUNDO_CUATRIMESTRE`. Además de la definición del tipo, queremos disponer de dos funciones que nos permitan convertir un valor de `TipoAsignatura` en una `Cadena` y viceversa:

- `void deTipoAsignaturaACadena(Cadena res, TipoAsignatura tipo)`. Almacena en el parámetro de salida `res` una cadena con la transcripción del valor contenido en el parámetro de entrada `tipo`. Por ejemplo, si `tipo` contiene el valor `PRIMER_CUATRIMESTRE`, debe guardarse la cadena "`PRIMER_CUATRIMESTRE`" en el parámetro de salida `res`.
- `int deCadenaATipoAsignatura(const Cadena tipo, TipoAsignatura *res)`. Almacena en el parámetro de salida `res` el valor adecuado de `TipoAsignatura`, en función del contenido de la cadena `tipo`. Por ejemplo, si `tipo` contiene la cadena "`PRIMER_CUATRIMESTRE`", se almacenará el valor `PRIMER_CUATRIMESTRE` en `res`. Si la cadena contenida en `tipo` no coincide con ninguno de los valores posibles de `TipoAsignatura`, la función devuelve -1, indicando con ello que se ha producido un error (en caso contrario, la función debe devolver 0).

EL TIPO ASIGNATURA

Se desea disponer de un tipo estructura `Asignatura`, con los siguientes campos:

- **nombre**, de tipo `Cadena`.
- **codigo**, de tipo `TipoCodigo`. Debe declarar previamente `TipoCodigo` como una cadena de texto que permita almacenar exactamente 7 caracteres.
- **creditos**, de tipo `double`.
- **tipo**, de tipo `TipoAsignatura`.
- **curso**, de tipo `int`.
- **departamento**, de tipo `Cadena`.

Declare también el tipo `PAsignatura` (puntero a `Asignatura`), y el tipo `ArrayAsignaturas` (array de `Asignatura` que permite almacenar hasta 100 asignaturas; tenga en cuenta que este valor podría cambiar en el futuro).

Además de las definiciones de tipos, queremos disponer de las siguientes funciones (tenga en cuenta que es posible que para resolver algunas de ellas sea necesario implementar funciones auxiliares):

- `int inicializaAsignatura(PAsignatura res, const Cadena nombre, const TipoCodigo codigo, double creditos, TipoAsignatura tipo, int curso, const Cadena departamento)`. Comprueba que los valores recibidos por parámetros cumplen con las restricciones del tipo `Asignatura`, y si es así, inicializa los campos de `res` con dichos valores. Si se incumple alguna de las restricciones de `Asignatura`, se debe informar de ello por pantalla, y la función devolverá -1 (en caso contrario, devuelve 0). Se considerarán las mismas restricciones para el tipo `Asignatura` que en entregas anteriores (relativas al código, los créditos y el curso).
- `void getAcronimo(Asignatura a, Cadena res)`. Devuelve en `res` el acrónimo de la asignatura `a`, formado por los caracteres mayúsculas del nombre de la asignatura.
- `int leeAsignaturaTeclado(PAsignatura res)`. Solicita al usuario que introduzca por teclado cada uno de los valores de las propiedades de una asignatura, e inicializa con dichos valores la asignatura apuntada por `res`. El tipo de la asignatura se introducirá mediante una cadena de

caracteres que coincidirá con uno de los valores posibles de `TipoAsignatura`. Si se incumple alguna restricción, o se introduce un valor no válido para el tipo de la asignatura, la función devuelve -1 (en caso contrario, devuelve 0).

- `void muestraAsignatura(Asignatura a)`. Muestra por pantalla cada una de las propiedades de la asignatura `a` (una propiedad por línea, poniendo el nombre de la propiedad seguido del valor de la propiedad en cada línea). A continuación se muestra un ejemplo de la visualización que se debe obtener:

```
Nombre: Fundamentos de Programación
Acrónimo: FP
Código: 2050001
Créditos: 12.0
Tipo: ANUAL
Curso: 1
Departamento: LSI
```

- `int leeAsignaturasTeclado(ArrayAsignaturas res)`. Solicita al usuario que introduzca por teclado un número positivo que representa el número de asignaturas que desea leer, y a continuación va leyendo las asignaturas una por una y almacenándolas en el array `res`. Si se introduce un valor no válido para el número de asignaturas a leer, se le indicará al usuario y se le volverá a pedir que introduzca dicho número (hasta que introduzca un valor correcto). Igualmente, si en alguna de las lecturas individuales de las asignaturas se produce algún error, se le volverá a solicitar al usuario que introduzca los datos de dicha asignatura. La función devuelve el número de asignaturas leídas.
- `void muestraAsignaturas(const ArrayAsignaturas res, int nAsig)`. Muestra por pantalla cada una de las asignaturas del array `res`, precedida cada una por un mensaje indicando el número de la asignatura que se está mostrando.
- `int leeAsignaturasFichero(const Cadena nombreFichero, ArrayAsignaturas res)`. Lee asignaturas desde un fichero de texto y las almacena en un array. Devuelve el número de asignaturas leídas, o 0 si no se encuentra el fichero. El fichero de texto tiene el formato que se indica más abajo, y se supone que está bien construido y contiene valores correctos para las propiedades de las asignaturas.

```
Fundamentos de Programación
1234567
12.0
ANUAL
1
LSI
. . .
```

EL TIPO GRADO

Se desea disponer de un tipo estructura `Grado`, con los siguientes campos:

- **nombre**, `de tipo Cadena.`
- **centro**, `de tipo Cadena.`
- **obligatorias**, `de tipo ArrayAsignaturas.`
- **numObligatorias**, `de tipo int.`
- **optativas**, `de tipo ArrayAsignaturas.`
- **numOptativas**, `de tipo int.`
- **minimoCreditosOptativas**, `de tipo double.`

Declare también el tipo `PGrado` (puntero a `Grado`).

Además de las definiciones de tipos, queremos disponer de las siguientes funciones (tenga en cuenta que es posible que para resolver algunas de ellas sea necesario implementar funciones auxiliares):

- `int inicializaGrado(PGrado res, const Cadena nombre, const Cadena centro, const ArrayAsignaturas obligatorias, int numObligatorias, const ArrayAsignaturas optativas, int numOptativas, double minimoCreditosOptativas)`. Comprueba que los valores recibidos por parámetros cumplen con las restricciones del tipo `Grado`, y si es así, inicializa los campos de `res` con dichos valores. Si se incumple alguna de las restricciones de `Grado`, se debe informar de ello por pantalla, y la función devolverá -1 (en caso contrario, devuelve 0). Se considerarán las mismas restricciones para el tipo `Grado` que en entregas anteriores (relativas a la propiedad `minimoCreditosOptativas` y al número de créditos de las asignaturas optativas).
- `void muestraGrado(Grado g)`. Muestra por pantalla cada una de las propiedades del grado `g`, incluyendo las propiedades de cada una de las asignaturas obligatorias y optativas del grado. A continuación se muestra un ejemplo de la visualización que se debe obtener:

```
Nombre: Grado en Ingeniería Informática
Centro: ETS Ingeniería Informática
Mínimo créditos optativas: 30
Asignaturas obligatorias:
***
Aquí vendría la salida de la función muestraAsignatura(…) para cada asignatura obligatoria
***
Asignaturas optativas:
***
Aquí vendría la salida de la función muestraAsignatura(…) para cada asignatura optativa
***
```

EL TIPO TIPOESPACIO

Se desea disponer de un tipo enumerado `TipoEspacio`, con los valores `TEORIA`, `LABORATORIO`, `SEMINARIO`, `EXAMEN` y `OTRO`. Además de la definición del tipo, queremos disponer de dos funciones que nos permitan convertir un valor de `TipoEspacio` en una `Cadena` y viceversa:

- `void deTipoEspacioACadena(Cadena res, TipoEspacio tipo)`. Almacena en el parámetro de salida `res` una cadena con la transcripción del valor contenido en el parámetro de entrada `tipo`. Por ejemplo, si `tipo` contiene el valor `TEORIA`, debe guardarse la cadena "`TEORIA`" en el parámetro de salida `res`.
- `int deCadenaATipoEspacio(const Cadena tipo, TipoEspacio *res)`. Almacena en el parámetro de salida `res` el valor adecuado de `TipoEspacio`, en función del contenido de la cadena `tipo`. Por ejemplo, si `tipo` contiene la cadena "`TEORIA`", se almacenará el valor `TEORIA` en `res`. Si la cadena contenida en `tipo` no coincide con ninguno de los valores posibles de `TipoEspacio`, la función devuelve -1, indicando con ello que se ha producido un error (en caso contrario, la función debe devolver 0).

EL TIPO ESPACIO

Se desea disponer de un tipo estructura `Espacio`, con los siguientes campos:

- **nombre**, de tipo `Cadena`.
- **planta**, de tipo `int`.
- **tipo**, de tipo `TipoEspacio`.
- **capacidad**, de tipo `int`.

Declare también el tipo `PEspacio` (puntero a `Espacio`), y el tipo `ArrayEspacios` (array de `Espacio` que permite almacenar hasta 200 espacios; tenga en cuenta que este valor podría cambiar en el futuro).

Además de las definiciones de tipos, queremos disponer de las siguientes funciones (tenga en cuenta que es posible que para resolver algunas de ellas sea necesario implementar funciones auxiliares):

- `int inicializaEspacio(PEspacio res, const Cadena nombre, int planta, TipoEspacio tipo, int capacidad)`. Comprueba que los valores recibidos por parámetros cumplen con las restricciones del tipo `Espacio`, y si es así, inicializa los campos de `res` con dichos valores. Si se incumple alguna de las restricciones de `Espacio`, se debe informar de ello por pantalla, y la función devolverá -1 (en caso contrario, devuelve 0). Se considerará la misma restricción para el tipo `Espacio` que en entregas anteriores (relativa a capacidad).
- `int leeEspacioTeclado(PEspacio res)`. Solicita al usuario que introduzca por teclado cada uno de los valores de las propiedades de un espacio, e inicializa con dichos valores el espacio apuntada por `res`. El tipo del espacio se introducirá mediante una cadena de caracteres que coincidirá con uno de los valores posibles de `TipoEspacio`. Si se incumple alguna restricción, o se introduce un valor no válido para el tipo del espacio, la función devuelve -1 (en caso contrario, devuelve 0).
- `void muestraEspacio (Espacio e)`. Muestra por pantalla cada una de las propiedades del espacio `e` (una propiedad por línea, poniendo el nombre de la propiedad seguido del valor de la propiedad en cada línea). A continuación se muestra un ejemplo de la visualización que se debe obtener:

  ```
  Nombre: A3.10
  Planta: 3
  Tipo: EXAMEN
  Capacidad: 210
  ```

- `int leeEspaciosTeclado(ArrayEspacios res)`. Solicita al usuario que introduzca por teclado un número positivo que representa el número de espacios que desea leer, y a continuación va leyendo los espacios uno por uno y almacenándolos en el array `res`. Si se introduce un valor no válido para el número de espacios a leer, se le indicará al usuario y se le volverá a pedir que introduzca dicho número (hasta que introduzca un valor correcto). Igualmente, si en alguna de las lecturas individuales de los espacios se produce algún error, se le volverá a solicitar al usuario que introduzca los datos de dicho espacio. La función devuelve el número de espacios leídos.
- `void muestraEspacios(const ArrayEspacios res, int nEspacios)`. Muestra por pantalla cada uno de los espacios del array `res`, precedido cada uno por un mensaje indicando el número del espacio que se está mostrando.
- `int leeEspaciosFichero(const Cadena nombreFichero, ArrayEspacios res)`. Lee espacios desde un fichero de texto y las almacena en un array. Devuelve el número de espacios leídos, o 0 si no se encuentra el fichero. El fichero de texto tiene el formato que se indica más abajo,

y se supone que está bien construido y contiene valores correctos para las propiedades de los espacios.

```
A3.10
3
EXAMEN
210
. . .
```

EL TIPO CENTRO

Se desea disponer de un tipo estructura `Centro`, con los siguientes campos:

- **nombre**, `de tipo Cadena.`
- **direccion**, `de tipo Cadena.`
- **numeroPlantas**, `de tipo int.`
- **numeroSotanos**, `de tipo int.`
- **espacios**, `de tipo ArrayEspacios.`
- **numEspacios**, `de tipo int.`

Declare también el tipo `PCentro` (puntero a `Centro`) y el tipo `ArrayInt` (array de 5 elementos de tipo `int`).

Además de las definiciones de tipos, queremos disponer de las siguientes funciones (tenga en cuenta que es posible que para resolver algunas de ellas sea necesario implementar funciones auxiliares):

- `int inicializaCentro(PCentro res, const Cadena nombre, const Cadena direccion, int numeroPlantas, int numeroSotanos, const ArrayEspacios espacios, int numEspacios).` Comprueba que los valores recibidos por parámetros cumplen con las restricciones del tipo `Centro`, y si es así, inicializa los campos de `res` con dichos valores. Si se incumple alguna de las restricciones de `Centro`, se debe informar de ello por pantalla, y la función devolverá -1 (en caso contrario, devuelve 0). Se considerarán las mismas restricciones para el tipo `Centro` que en entregas anteriores (relativas a las propiedades `numeroPlantas` y `numeroSotanos`).
- `void muestraCentro (Centro c).` Muestra por pantalla cada una de las propiedades del centro `c`, incluyendo las propiedades de cada uno de los espacios del centro. A continuación se muestra un ejemplo de la visualización que se debe obtener:

  ```
  Nombre: ETS Ingeniería Informática
  Dirección: Avda Reina Mercedes s/n
  Número de plantas: 5
  Número de sótanos: 1
  Espacios:
  ***
  Aquí vendría la salida de la función muestraEspacio(…) para cada espacio del centro
  ***
  ```
- `void getConteosEspacios(const ArrayEspacios espacios, int nEspacios, ArrayInt c).` Construye un array de 5 elementos de tipo `int` que representan el número de espacios de tipo aula de teoría, laboratorio, seminario, aula de examen u otro tipo, respectivamente, que hay en el centro.

TEST

Pruebe todas las funciones que ha implementado. Para ello, cree los ficheros `testCadena.c`, `testTipoAsignatura.c`, `testAsignatura.c`, `testGrado.c`, `testTipoEspacio.c`, `testEspacio.c` y `testCentro.c`. Cada uno de ellos contendrá una función `main` que irá ejecutando los casos de prueba necesarios para comprobar el correcto funcionamiento de cada una de las funciones definidas en el tipo correspondiente.

Para compilar el proyecto vaya a *Project* > *Build Project*. Se creará un fichero ejecutable por cada test que no contenga errores (es posible que tenga que recargar la ventana del explorador de proyectos para que se muestren).

Para lanzar un test, colóquese encima del ejecutable, pulse el botón derecho del ratón y en el menú contextual seleccione *Run As* > *Local C/C++ Project*. La salida del test se mostrará en la consola.