Técnicas y prácticas de programación

Directivas

Directivas del preprocesador

Directivas: Instrucciones al compilador antes de que se ejecute el programa principal. Las más usuales son **#include** y **#define**.

¿Para qué sirve la directiva #include?

#include indica al compilador que lea el archivo fuente y que inserte su contenido en la posición donde se encuentra dicha directiva

Directiva #include

```
#include <stdio.h>
#include <stdlib.h>
#include "primo.h"
```

Directorio por defecto include

Directorio actual

Archivos .h

Prototipo de las funciones

Los prototipos están formados por el tipo de dato que retorna la función, el nombre de la función y para cada parámetro su tipo y nombre (el nombre de los parámetros es opcional). Para que este completo debe terminar en punto y coma .

```
int unidades(int);
```

Usar diferentes funciones facilita la modularización del programa, facilita el reúso, la cohesión y el bajo acoplamiento

Declarar vs implementar una operación

```
int esPrimo(int numero);
```

Declarar

```
// retorna 0 si no es primo y 1 si sí lo es
int esPrimo(int numero){
    int cont = 0, i=1;
    while (i <= numero){
        if (numero % i == 0){
            cont++;
        }
        i++;
    }
    if (cont == 2){
        return 1;
    }else{
        return 0;
    }
}
```

Implementar

Declarar: Indicar el prototipo de la operación

Implementar: Escribir el cuerpo de la operación

La implementación de la operación y el llamado debe coincidir con lo indicado en el prototipo

Headers - archivos .h

Definen prototipos de funciones, constantes y tipos de datos.

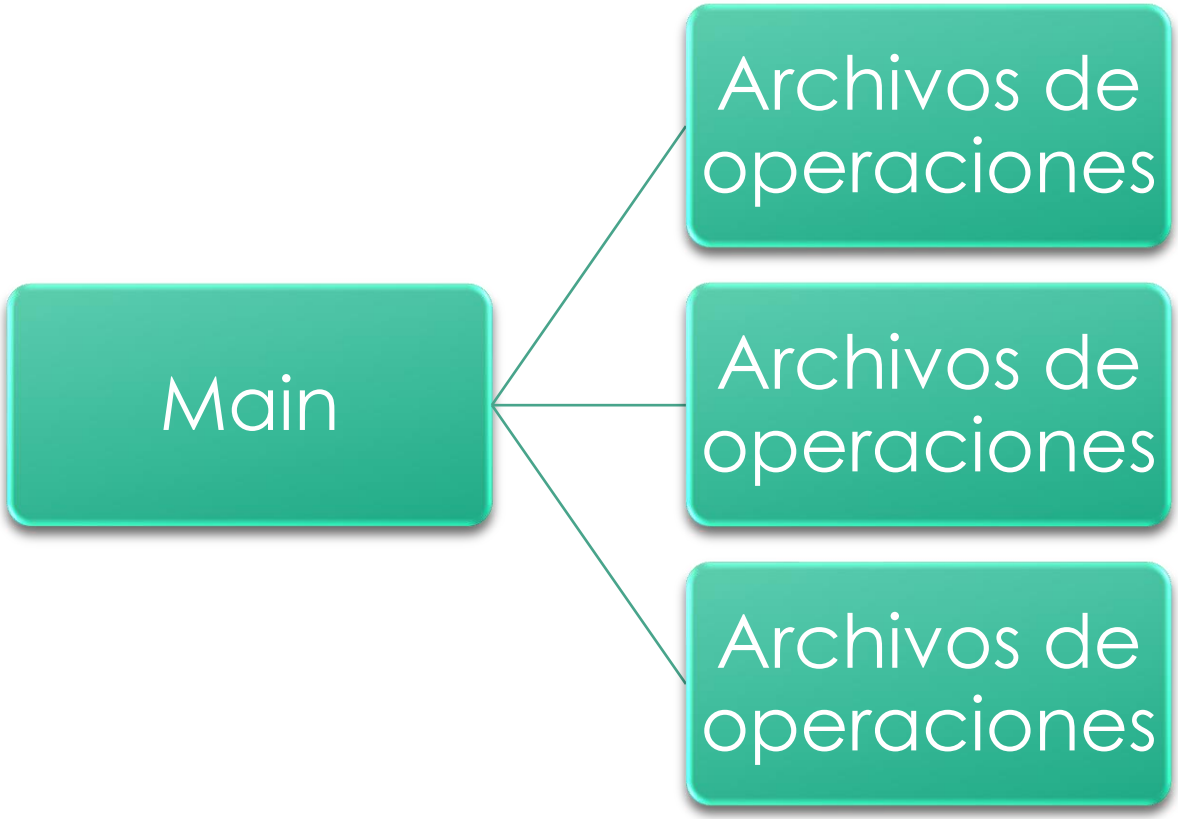
Se usan como "librería" para usar las operaciones definidas en los prototipos en diferentes archivos .c

Permite separar un programa en diferentes archivos

¿Cómo llevar a cabo la separación?

La función **main** debe conocer que existen algunas funciones que están definidas en otros archivos. Para ello, se definen los prototipos de las funciones y estos se incluyen en archivos con extensión .h

Separación de programas en varios archivos



Se requiere Incluir los archivos en el programa principal. (El archivo que tiene el main)

Headers (.h) y archivos fuente (.c)

.h: Contendrán los prototipos de las operaciones

.c: Contendrán la implementación de las operaciones declaradas en los .h

Ejemplo separación

Primo.c

```c
#include "primo.h"

// retorna 0 si no es primo y 1 si sí lo es
int esPrimo(int numero){
    int cont = 0, i=1;
    while (i <= numero){
        if (numero % i == 0){
            cont++;
        }
        i++;
    }
    if (cont == 2){
        return 1;
    }else{
        return 0;
    }
}
```

Ejemplo separación

Primo.h

```c
#include <stdio.h>
#include <stdlib.h>
int esPrimo(int numero);
```

Main.c

```c
#include "primo.h"

int main()
{
    int numero, resultado;
    printf("Hola Indique un numero para ver si es primo: \n");
    scanf("%d",&numero);
    resultado = esPrimo(numero);
    if (resultado == 0){
        printf("El numero %d no es primo\n",numero );
    }else{
        printf("El numero %d si es primo\n",numero );
    }
    system("PAUSE");
    return 0;
}
```

¿Cómo se genera un ejecutable en C?

Preprocesamiento	Compilación	Enlazamiento
Reemplaza las instrucciones del preprocesa-dor para dejar el código listo para compilación	Transforma el código al lenguaje ensamblador propio de la máquina en la que se compila (extensión .s) y traduce de lenguaje ensamblador a código binario entendible por el procesador de la máquina (extensión .o	Se incorpora el código binario de los diferentes archivos en un solo ejecutable

Compilación

gcc **–c** Primo.c (compilar archivo fuente)

gcc **–c** Main.c (compilar archivo fuente)

gcc Primo.o Main.o **–o** ejecutable(enlazar y general ejecutable)

-c: pre-procesamiento, compilación, ensamblado. Genera código objeto con extensión .o (de ensamblador a binario).

-o: pre-procesamiento, compilación, ensamblado, enlace. **Sólo enlaza si se pasan archivos con extensión .o**

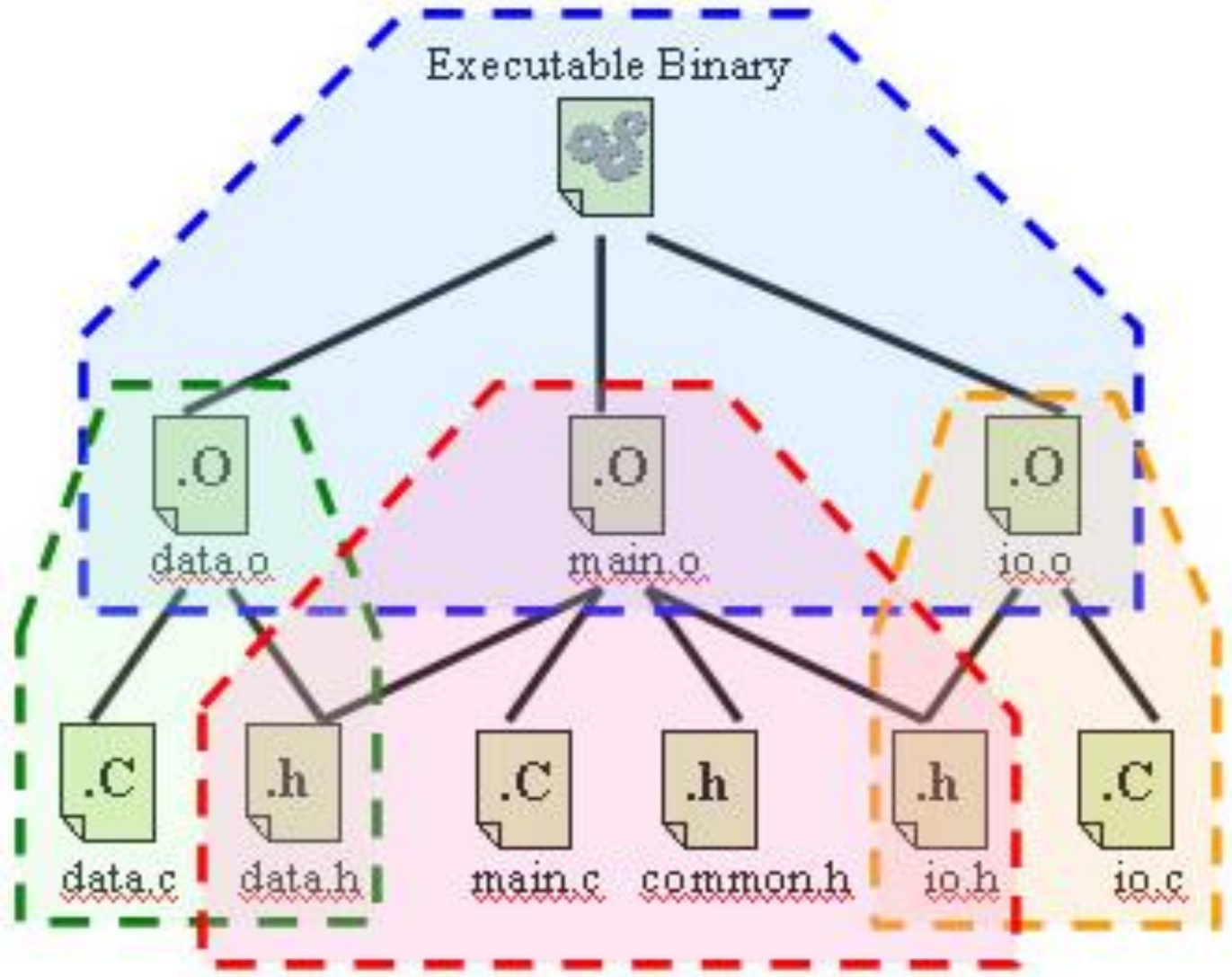
Makefiles

¿Cuáles son las reglas básicas de separación del código?

Los ficheros .h deben contener sólo los prototipos de las operaciones de los programas

Los archivos .c contienen las implementaciones de los prototipos

¿Cómo se relacionan las dependencias de un proyecto?



https://paragasu.files.wordpress.com/2008/10/makefile.jpeg

¿Cómo se compilan diferentes archivos ?

Un programa puede estar formado por muchos archivos fuente. Compilarlos de manera independiente es una tarea tediosa y propensa a errores.

¿Qué archivos fueron modificados?

¿Cómo se llaman cada uno de esos archivos?

Existe un programa que automatiza todo el proceso

Herramienta GNU Make

GNU Make es una herramienta para actualizar, en forma optimizada y automática, los diversos archivos de programas que integran un proyecto de software. Esta formado por reglas

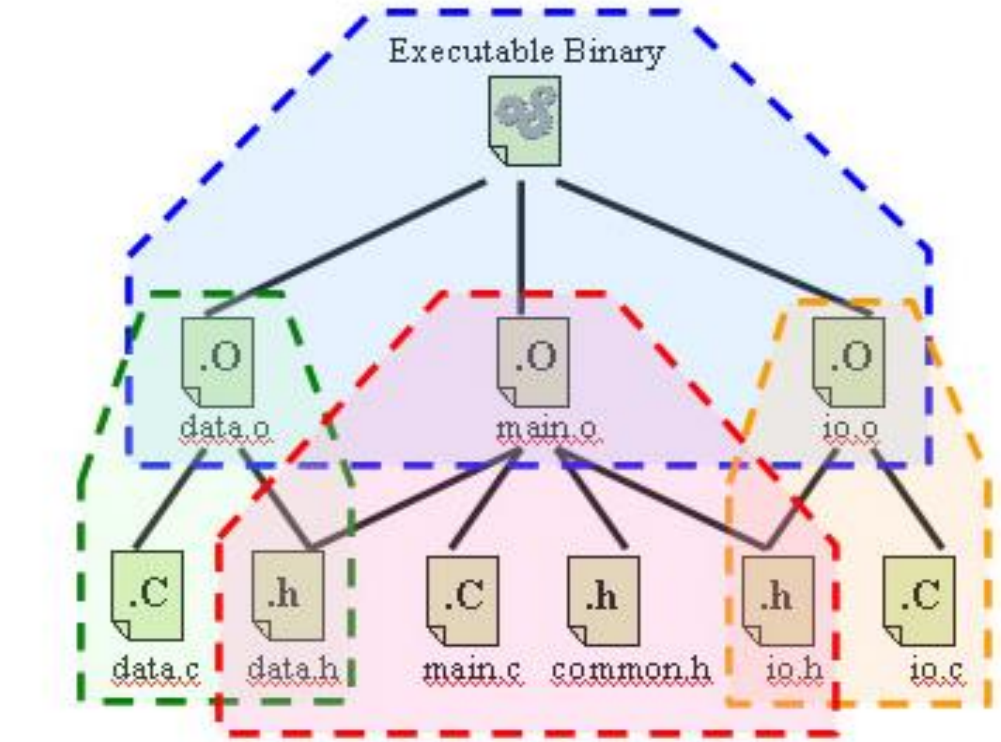
¿Cómo se usa la herramienta GNU Make?

Se construye un **makefile.** Un archivo de texto (sin extensión) que contiene las reglas para cumplir un objetivo (compilar el programa). Si hay modificaciones, sólo se recompila los archivos modificados y los programas que dependen del archivo modificado.

Un ejemplo de un makefile

```
all: programa
programa: Primo Main
	gcc -o Ejecutable Main.o Primo.o
Main: Main.c
	gcc -c Main.c
Primo: Primo.c
	gcc -c Primo.c
```

¿Cómo se relacionan los makefiles con los arhivos que se compilan?



https://paragasu.files.wordpress.com/2008/10/makefile.jpeg

```
all: data.o io.o main.o
	gcc data.o io.o main.o -o myexecutable
data.o: data.c data.h
	gcc -c data.c
io.o: io.c io.h
	gcc -c io.c
main.o: main.c data.h io.h common.h
	gcc -c main.c
```

¿Cuál es el formato de un archivo makefile?

Comentarios

- Facilita mejor entendimiento de las reglas (sintaxis #)

Variables

- Simplifica la creación del archivo. Declaración nombre = dato. Luego *$(nombre)* para usarla.

Reglas

- Indican qué archivos dependen de otros archivos y qué comandos se requieren para compilar un archivo en particular

¿Cuál es el formato de un archivo makefile?

Comentarios

- `# Este es un comentario`

Variables

- `SRC = main.c`
- `gcc $(SRC)`

Reglas

- `main: main.c funciones.h`
- `gcc -o main main.c funciones.h`

¿Cómo hacer funcionar la herramienta make en windows?

- Copiar el archivo c:\MinGW\bin\mingw32-make
- Pegarlo y renombrarlo como ***make.exe***
- ***Ruta final :*** c:\MinGW\bin\make.exe
- **Configurar esta ruta en el path del sistema operativo**
- Si todo salió bien esto es lo que muestra

```
PS E:\Dropbox\PERSONAL\JAVERIANA\CURSOS\TECNICAS Y PRACTICAS\2020-2\pruebas> make
make: *** No targets specified and no makefile found.  Stop.
PS E:\Dropbox\PERSONAL\JAVERIANA\CURSOS\TECNICAS Y PRACTICAS\2020-2\pruebas>
```

Ejecutar el archivo make file

Dentro del directorio en el que crearon el makefile escribir en consola

```
make
```

Si el nombre del archivo no es makefile se puede especificar así

```
make –f nombreArchivo
```

```
gcc -c Primo.c
gcc -c Main.c
gcc -o Ejecutable Main.o Primo.o
```

Actividad en clase

Partiendo de los archivos:

Primos.c Primos.h y Main.c

1.Modifique los archivos y las líneas de código correspondientes para que ahora se llamen:

Tools.c Tools.h Main.c

2.Cree en sus archivos de herramientas (Tools) adicional a la de saber si un número es primo, una nueva operación que ayude a saber cuál es el MCD de 2 números positivos. (Recuerde que el MCD de 2 números, es el número más grande que logra dividir exactamente a ambos números, Ej: de 12 y 8 sería 4)

3.En su Main, haga un pequeño menú para que el usuario pueda elegir entre las 2 herramientas que su programa brinda (Conocer si es primo, y hallar el MCD de 2 números).

4. Haga uso de un makefile para la compilación y enlazamiento de sus archivos.