Laboratorio de PRO2

Sesión 1: entorno de trabajo y repaso de C++

Objetivos de la sesión

- Presentar el entorno de trabajo del laboratorio.
- Repasar la compilación y ejecución de programas en C++.
- Introducir el uso básico de `Makefile`.
- Recordar la estructura habitual de un programa en C++.
- Practicar acciones, funciones y paso de parámetros.

Entorno de trabajo

- Trabajaremos en **Linux**.
- El lenguaje de programación será **C++**.
- Usaremos el compilador g++.

Información de la asignatura

`https://www.cs.upc.edu/~pro2/`

- Deberíais tener una carpeta para la asignatura de Pro2.
- Es recomendable crear una carpeta distinta para cada sesión.

Compilar un programa sencillo

Partimos de un fichero fuente de C++:

`PRO2.cc`

Para obtener el ejecutable:

```
g++ -o PRO2.exe PRO2.cc
```

- `-o` permite escoger el nombre del ejecutable.
- Si no hay errores y existe `main`, se genera `PRO2.exe`.

Compilación y enlazado

En programas más grandes suele convenir compilar por separado:

```
g++ -c PRO2.cc
g++ -o PRO2.exe PRO2.o
```

Caso general:

```
g++ -c f_main.cc f1.cc ... fn.cc
g++ -o f_main.exe f_main.o f1.o ... fn.o
```

Idea clave

Primero se generan ficheros objeto `.o`; después se enlazan para formar el ejecutable.

Opciones de compilación del Jutge

Conviene compilar con las mismas opciones que usará el Jutge:

```
-g3 -D_GLIBCXX_DEBUG -Wall -Wextra -Werror \
-Wno-sign-compare -std=c++17 \
-fno-extended-identifiers
```

- `-Wall -Wextra`: activa muchos avisos del compilador.
- `-Werror`: trata los avisos como errores.
- `-D_GLIBCXX_DEBUG`: ayuda a detectar accesos incorrectos a contenedores estándar.

Makefile básico

Un fichero `Makefile` evita escribir siempre los mismos comandos. Se ejecuta escribiendo make.

```
CXX := g++
CXXFLAGS := -g3 -D_GLIBCXX_DEBUG -Wall -Wextra \
            -Werror -Wno-sign-compare -std=c++17 \
            -fno-extended-identifiers

PRO2.exe: PRO2.o
        $(CXX) $(CXXFLAGS) -o PRO2.exe PRO2.o

PRO2.o: PRO2.cc
        $(CXX) $(CXXFLAGS) -c PRO2.cc

clean:
        rm -f PRO2.exe PRO2.o

.PHONY: clean
```

Ejecutar programas

Ejecución interactiva:

```
./PRO2.exe
```

Redirección de entrada: los datos de entrada se toman de PRO2.dat

```
./PRO2.exe < PRO2.dat
```

Redirección de entrada y salida: los datos de entrada se toman de PRO2.dat y los datos de salida se escriben en PRO2.sal

```
./PRO2.exe < PRO2.dat > PRO2.sal
```

Idea clave

La entrada y la salida estándar pueden venir de teclado(entrada)/pantalla(salida) o de ficheros.

Estructura de un programa C++

Un programa sencillo suele tener esta organización:

1. Inclusiones y espacio de nombres.
2. Definiciones de constantes y tipos.
3. Acciones y funciones auxiliares.
4. Programa principal: `main`.

Inclusiones típicas:

- `#include <iostream>`
- `#include <vector>`
- `#include <string>`
- `#include <cmath>`

Acciones, funciones y parámetros

Tipos de parámetros en una acción:

- **Entrada**: el valor inicial no debe cambiar. El valor se usa, pero no se tiene que modificar.
- **Entrada/salida**: el valor inicial importa y puede modificarse.
- **Salida**: el valor final lo crea la acción.

Traducción habitual a C++:

- Por valor, sin & ni const: valor simple (int, string, bool, char) y de entrada.
- Por referencia, &: valores entrada/salida o salida.
- Por referencia constante, `const T&`: para valores de entrada que son objetos grandes. Pasamos su "dirección"para no hacer una copia de un objeto grande, pero prohibimos su modificación.

Ejemplos de cabeceras

Intercambio de dos enteros:

```
void intercambiar(int& m, int& n)
```

En intercambiar, m y n son de entrada/salida. Ponemos &.

Intercambio de dos posiciones de un vector:

```
void intercambiar_vect(vector<int>& v, int i, int j)
```

En intercambiar_vect, v es de entrada/salida y ponemos &. i y j son de entrada, solo nos importa el valor que tienen al inicio, y no se modificarán. Son valores simples. No usan ni & ni const.

Búsqueda lineal en un vector:

```
bool busqueda_lin(const vector<int>& v, int x)
```

En busqueda_lin, v y x son valores de entrada unicamente. Como x es simple, no ponemos ni const ni &. v no es simple, y por tanto lleva const y &.

Paso de parámetros: cuadro resumen

	Valores de entrada	**Valores de entrada/salida**
Valores simples	`ni const ni &` Por valor. Ej.: `busqueda_lin` Parámetro: `x` (`int x`)	`&` Por referencia. Ej.: `intercambiar` Parámetros: `m,n` (`int& m, int& n`)
Objetos grandes	`const y &` Referencia constante. Ej.: `busqueda_lin` Parámetro: `v` (`const vector<int>& v`)	`&` Referencia modificable. Ej.: `intercambiar_vect` Parámetro: `v` (`vector<int>& v`)

Ejercicios de la sesión

- Suma de una secuencia de enteros.
- Intercambio de dos variables enteras.
- Búsqueda lineal en un vector de enteros.
- Intercambio de dos posiciones de un vector.
- Máximo y mínimo de un vector.
- Suma y producto de matrices.
- Clasificación de una liga.