@prog2ua

# Errores de compilación y ejecución

## Programación 2

Grado en Ingeniería Informática
Universidad de Alicante
Curso 2024-2025

# Compilación (1/3)

- Orden para compilar un programa fuente `hello.cc`:

Terminal

```
$ g++ -Wall -g hello.cc -o hello
```

  - `-Wall`: activar todos los *warnings*
    - Un *warning* es un aviso de que algo podría estar mal, aunque no es del todo incorrecto
    - Un buen programador debería eliminar todos los *warnings*
  - `-g`: añade información para depuración al programa ejecutable
  - `-o hello`: indica que el ejecutable generado debe llamarse `hello`

- Si no hay errores ni *warnings*, el compilador genera el ejecutable sin mostrar ningún mensaje

# Compilación (2/3)

- ¿Y si hay errores?

```
int main(){
  cout << "Hello, world" << endl // Falta el ';' final
  return 0;
}
```

Terminal

```
hello.cc: In function ‘int main()’:
hello.cc:10:3: error: expected ‘;’ before ‘return’
   return 0;
   ^
```

- Traducción del mensaje:
  - Hay algo mal en la línea 10, columna 3 (o antes)
- “El compilador está en inglés y no entiendo el mensaje”
  - Solución: aprende inglés (puedes usar el traductor de Google mientras tanto)

# Compilación (3/3)

- ¿Y si hay muuuuchos errores?
  - Si no es posible verlos bien en pantalla, hay que guardarlos en un fichero para verlos bien:

Terminal

```
$ g++ -Wall -g hello.cc -o hello 2> errors.txt
```

  - A veces, un error en el código provoca otros errores en las líneas siguientes (errores *en cascada*), porque el compilador no puede recuperarse del error. ¿Qué hay que hacer?
    1. Corregir el primer error
    2. Volver a compilar

# Errores de ejecución (1/4)

- En Programación 2 utilizaremos *Valgrind* para la detección de algunos errores de ejecución
- *Valgrind* es un *memory checker*, un programa que controla los accesos a memoria de otro programa, buscando errores en la gestión de memoria dinámica, accesos incorrectos a vectores y matrices, uso de variables sin inicializar, etc.
- Ralentiza la ejecución del programa, pero detecta muchos errores (aunque no todos)
- Igual que sucede al compilar, es posible que *Valgrind* detecte muchos errores, en cuyo caso hay que redirigir los mensajes a un fichero:

Terminal

```
$ valgrind -q hello 2> errors.txt
```

- La opción `-q` es para que *Valgrind* solamente muestre errores

# Errores de ejecución (2/4)

- Ejemplo 1: acceso incorrecto a un vector

```
int v[]={0,1,2,3};
int suma=0;
for(unsigned i=0;i<=4;i++){ // Error: debe ser i<4
  suma+=v[i]; // v[4] está fuera de 'v'
}
cout << suma; // Valgrind da error cuando se usa 'suma'
```

```
$ valgrind -q ./ej1
==24554== Conditional jump or move depends on uninitialised value(s)
==24554==    at 0x4E87B83: vfprintf (vfprintf.c:1631)
==24554==    by 0x4E8F898: printf (printf.c:33)
==24554==    by 0x400600: main (ejemplo1-valgrind.cc:15)
==24554==
==24554== Use of uninitialised value of size 8
==24554==    at 0x4E8476B: _itoa_word (_itoa.c:179)
==24554==    by 0x4E8812C: vfprintf (vfprintf.c:1631)
==24554==    by 0x4E8F898: printf (printf.c:33)
==24554==    by 0x400600: main (ejemplo1-valgrind.cc:15)
==24554==
==24554== Conditional jump or move depends on uninitialised value(s)
==24554==    at 0x4E84775:  itoa word ( itoa.c:179)
```

- El error se detecta porque `v[4]` está *marcado* como memoria sin inicializar, de ahí el mensaje que sale

# Errores de ejecución (3/4)

- Ejemplo 2: uso de variable sin inicializar

```
int j,k;
k=j+7;
if(k<10)
  cout << "j es menor que 3" << endl;
```

```
$ valgrind -q ./ej2
==24603== Conditional jump or move depends on uninitialised value(s)
==24603==    at 0x40085B: main (ejemplo2-valgrind.cc:13)
==24603==
j es menor que 3
```

- Si se usa `-Wall` al compilar, se puede detectar algún fallo de estos:

```
$ g++ -o ej2 -g -Wall ejemplo2-valgrind.cc
ejemplo2-valgrind.cc: In function ‘int main()’:
ejemplo2-valgrind.cc:11:13: warning: ‘j’ is used uninitialized in this function [-Wuninitialized]
    k = j + 7;
            ^
```

# Errores de ejecución (4/4)

- Ejemplo 3: *Valgrind* detecta muchos errores, pero no todos

```
int main(){
  int v[4];
  for(unsigned i=0;i<=10;i++){
    v[i]=i;
  }
  cout << v[4];
}
```

```
$ valgrind -q ./ej3
4
*** stack smashing detected ***: ./ej3 terminated
Abortado (`core' generado)
```

- En este caso falla porque al salirse (escribiendo) de `v` se borran datos importantes de la función `main` en memoria