Tema 1: Organización de la memoria.

Diapositivas de clase.

- Aquí puedes descargar las diapositivas utilizadas en clase, que prácticamente contienen lo mismo que la web: diapositivas tema 1

Breve repaso de C.

- Punteros.
- Recursividad.
- Modificadores `extern`, `static`, `volatile`.
- Guarda de ficheros `.h`:

```
#ifndef XXX_H
#define XXX_H
  /* Código C */
#endif
```

  Puedes ver en este enlace un ejemplo de lo que tarda en compilar un proyecto real (LibreOffice) escrito en su mayoría en `C++`. Imagina si estuviera *plagado* de `includes` gratuitos!

- No tienen que ver con **C** pero te será muy útil saber usar herramientas como make, gdb o valgrind.

El lenguaje de programación C++.

Historia.

- El 14 de octubre de 1985 se publica cfront `1.0`, aquí puedes ver su código fuente y aquí puedes leer un artículo donde su creador (Bjarne Stroustrup) habla de los orígenes del lenguaje.
- Curiosamente `cfront` era un **traductor** de `C++` a `C`.
- El compilador g++ de GNU fue el primero en generar código nativo y no traducir a `C`. Aunque no todo el mundo está de acuerdo y otras fuentes citan al compilador Zortech C++ como el primero en hacerlo.

Importante.

- Es una extensión (más) *orientada a objetos* de `C`.
- Trata de ser compatible con `C` (pero no todo programa en `C` es un programa correcto en `C++`).
- Toma ideas del lenguaje Simula.
- Los principios en los que se basa son los de **eficiencia** y **flexibilidad**.
- Además de la POO añade más paradigmas como el de la programación genérica.
- Para conocer mejor la historia del origen de `C++` es recomendable que leas estos dos artículos.

Características añadidas a *C*.

- Clases y objetos.
- Espacios de nombres.
- Constructores, destructores y funciones *amigas*.
- Zonas de visibilidad (`public`, `private` y `protected`).
- Operador de resolución de ámbito (**::**).

- Entrada/Salida mejorada.
- Sobrecarga de funciones (mangling) y de operadores.
- Genericidad de funciones y de clases.
- Herencia (*simple* y *múltiple*), funciones virtuales.
- Nombres de operadores lógicos: `and` , `or` , `not` , `xor` , etc..., además de los que ya conoces de **C** (`&&` , `||` , `!` , `^` ...).
- Literal *puntero nulo* : `nullptr`
- Evaluación de expresiones en tiempo de compilación: constexpr.
- Antes de usar (*llamar*) una función el compilador debe conocer su prototipo o su definición.
- Valores por defecto en parámetros de funciones.
- No necesitaremos `typedef` de *struct* o *class*.
- Tipo cadena. Es una clase de la biblioteca estándar de C++, no es un tipo base del lenguaje.
- Excepciones para el tratamiento de errores.

Ejemplo de-todo-un-poco.

```
class TDerived : public TBase1, private TBase2, protected TBase3 {
public:  // Parte publica ----------------------------------------------
 TDerived (int a, std::string s = "") : TBase1 (a) , _s(s) { this->_a = a; _n++; } // c
 ~TDerived () { _s = ""; }     // destructor

 inline bool isOk () { return (_s != ""); } // inline explícito

 bool isOk (const std::string& s) {    // inline implícito
   if (not checkMe ()) return false;   // acceso a la parte privada, nombres de op.
   return (_s != s);                   // acceso a la parte privada
```

```
 friend std::ostream& operator<< (std::ostream& out, const TDerived& d) { // función am
   return out << d._a << '/' << d._s ; // necesita acceso a parte privada
 }

 // Redefinición operador suma entre dos objetos de clase TDerived
 TDerived operator+ (const TDerived& rhs) { return TDerived (_a + rhs._a, _s + rhs._s);

private:  // Parte privada ----------------------------------------
 bool checkMe () { return (_a > 0); } // método de instancia privado

 int _a;                      // datos de instancia
 std::string _s;              // datos de instancia
 static int _n;               // datos de clase
}

TDerived d1(1, "example"), d2(2);
d1 = d1 + d2;
```

Iso C++.

- `C++` no es un lenguaje creado por una compañía particular que de alguna manera controla como evoluciona.
- Su desarrollo está gestionado por un comité dentro del estándar ISO.
- A este estándar se han adherido diversos fabricantes de compiladores de C++ entre otros.
- Hay varias versiones publicadas del mismo, nos referimos a ellas por el año de publicación: `C++98`, `C++11`, `C++14`, `C++17`, `C++20` y `C++23`. Además está en desarrollo `C++26`.
- El comité se ha propuesto publicar una nueva versión cada tres años del estándar del lenguaje.
- Debemos comprobar la documentación del compilador usado para saber qué versión o versiones implementa y en este último caso elegir con cual trabajar.

" `--std=c++17` ", a `C++20` : " `--std=c++20` ", o a `C++23` : " `--std=c++23` ".

Enlaces interesantes sobre **C++**.

- Estándar ISO
- Cpp Reference
- The C++ Annotations
- Adaptación de `g++` a los distintos estándares.

Gestión y uso de la memoria.

Pila — *Stack* — (I)

- Es la memoria en la que se guardan variables locales y parámetros de funciones.
- Crea un ámbito automático. Es útil pero requiere tener cuidado:

```
int* suma_elementos (int n[5]) {
  static bool b = true;
  int s = 0;
  for (int i = 0; i < 5; i++) s += n[i];
  b = false;
  return &s;
}
```

- Por defecto los parámetros se pasan por valor:

```
int a = 0;                    |   int a = 0;
void inc (int p) { p++; }     |   void inc (int& p) { p++; }
inc (a);                      |   inc (a);
// BOOM!                      |   // OK ahora!
assert (a == 1);              |   assert (a == 1);
```

```
{                       // Ambito I
 int a = 2;
 {                      // Ambito II
   int b = 4
   int a = b;           // No es un error
 }
 assert ( a == 2 );     // Ok
}
```

- `C++` permite distinguir una variable global de una local que se llame igual con el operador de resolución de ámbito (**::**):

```
int a = 7;
int f () {
  int a = 3;
  return a + ::a;      // 10 = 3 + 7
}
```

Pila — *Stack* — (III)

- La llave de apertura ' `{` ' marca el inicio de un ámbito y la de cierre ' `}` ' el final.
- Cualquier variable local a un ámbito, al terminar el mismo, se destruye.
- ...pero esto es algo que ya conoceis de **Programación-I**. `C++` se comporta igual que `C` en este aspecto.

Pila — *Stack* — (IV)

- Hay que llevar especial cuidado con los punteros:

```
int main () {
  char* c = new char[20];
  return 0;
}
```

```
==7177== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==7177== Command: ./a.out
==7177==
```

- Continúa la salida de valgrind:

```
==7177== HEAP SUMMARY:
==7177==     in use at exit: 72,724 bytes in 2 blocks
==7177==   total heap usage: 2 allocs, 0 frees, 72,724 bytes allocated
==7177==
==7177== LEAK SUMMARY:
==7177==    definitely lost: 20 bytes in 1 blocks
==7177==    indirectly lost: 0 bytes in 0 blocks
==7177==      possibly lost: 0 bytes in 0 blocks
==7177==    still reachable: 72,704 bytes in 1 blocks
==7177==         suppressed: 0 bytes in 0 blocks
==7177== Rerun with --leak-check=full to see details of leaked memory
==7177==
==7177== For counts of detected and suppressed errors, rerun with: -v
==7177== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Almacenamiento global.

- Sigue las mismas reglas que en `c`.
- Una variable declarada fuera de cualquier función se considera que está en el almacenamiento global.
- La biblioteca estándar de **C++** usa el *espacio de nombres* **std**.
- El ámbito de vida de estas variables es el del programa.
- En el modelo multi-hilo de C++ las variables globales se comparten por todos los hilos.

Almacenamiento dinámico (*Heap*).

- Gestionado por el lenguaje con nuevos operadores:

  `new` , `new[]` , `delete` , `delete[]` .

```
...
class TArbol {...};
TArbol* a1 = new TArbol;
TArbol* a2 = new TArbol (a1);
delete a1; delete a2;
...
TArbol* va = new TArbol[10];
delete[] va;
```

- Se pueden sobrecargar a nivel global y por clases. En este vídeo del canal de Dave Plummer puedes obtener más información al respecto. También te recomiendo que veas su vídeo donde explica los problemas con la gestión de memoria que pueden ocurrir con las funciones de `c` para el tratamiento de cadenas.

¿Sabías que...?

- El operador `new` se puede *simular* en `c` con ayuda del preprocesador:

```
#define NEW(tipo,cantidad)  ((tipo*) malloc((cantidad) * (size_t) sizeof(tipo)))

/* 1: array de 20 caracteres */
char* c = NEW(char, 20);  // Equivale a:
                          // char* c = (char*) malloc (20 * sizeof(char))

/* 2: Matriz de 20*10 enteros */
int** m;
m = NEW(int*, 20);
for (int r = 0; r < 20; r++) {
  m[r] = NEW(int, 10);
  for (int c = 0; c < 10; c++)
    m[r][c] = 2*c;
}
```

¿Y sabías que también podemos...?

```
using namespace std;

void hola(void) {
  cout << "Hola\n";
}

void adios(void) {
  cout << "Adios\n";
}

using fpvv_t = void(*)(void);

int main()
{
  fpvv_t f;

  f = hola;
  f();

  f = adios;
  f();

  return 0;
}
```

Introducción a los TADs.

Definición.

- Un TAD Es un modelo matemático...
- ...de los objetos constituyentes de un tipo de datos, junto con...
- ...las funciones que operan sobre estos objetos.
- **IMPORTANTE**: estas funciones deben formar parte de la especificación del TAD.
- Un ejemplo: el TAD `Conjunto` puede ser definido como la colección de datos que son accedidos por operaciones como la `union`, `interseccion` y `diferencia`.

- Un TAD no es una *Estructura de Datos*. Esta última no es más que una colección de variables en un programa que están conectadas de una manera específica.
- En el diseño y especificación de un TAD nunca se habla de su representación interna. Esto permite usar eficientemente los principos de *ocultación de información* y de *reusabilidad*.
- Conseguir una especificación completa y rigurosa de un TAD implica obtener un conjunto de axiomas que lo describen completa e inequivocamente. Esto pude llegar a ser difícil con TADs complejos.

TADs estudiados.

- Listas simple y doblemente enlazadas.
- Pilas
- Colas
- Árboles
- Grafos

Aclaraciones.

- **Este contenido no es la bibliografía completa de la asignatura**, por lo tanto debes estudiar, aclarar y ampliar los conceptos que en ellas encuentres empleando los enlaces web y bibliografía recomendada que puedes consultar en la página web de la ficha de la asignatura y en la web propia de la asignatura.

la asignatura

Colas.