Clase #3
Introducción a los Paradigmas de la Programación

Paso de C a C++

- La principal diferencia entre C y C++ es que C++ está diseñado para programar objetos.
- C tiene sus propias bibliotecas de funciones: stdio.h, conio.h, string.h, math.h todas orientadas a la programación modular (imperativa).
- C++ también tiene sus propias bibliotecas, este conjunto es mejor conocido como la biblioteca estandar ó STL.

Paso de C a C++

- El esquema de programación procedural permite resolver los problemas desde un paradigma más simple: relaciones entre funciones.
- El esquema de programación por objetos es un paradigma que permite la abstracción de los problemas desde otra perspectiva: La especificación de un objeto.

Paso de C a C++

- Según Stroustroup:
  - Un lenguaje de programación sirve para dos cosas: proveer un mecanismo para expresar acciones a ser ejecutadas y proveer conceptos que serán utilizados cuando se piense que podrá ser hecho.
  - C es un lenguaje apegado a la máquina, para que sus recursos puedan ser utilizados fácilmente.
  - C++ es un lenguaje apegado al problema para que pueda ser resuelto con el “mínimo” esfuerzo.

Paso de C a C++

- C fue escogido como base de C++ porque:
  - Es versatil y relativamente de bajo nivel. Esto hace que C++ también sea utilizado en la programación sistema.
  - Corre donde sea y sobre cualquier cosa.
  - Encaja en el ambiente de programación UNIX.

Sugerencias para Programadores en C

- Los Macro son casi nunca utilizados. Utilice *const* o *enum* para las constantes, *inline* para lidiar con el overhear de las llamadas, *templates* para hacer familias de funciones.
- No declare una variable antes de que la necesite.
- No utilice *malloc, free*. El operador *new* y delete hacen el mismo trabajo y mejor.
- Trate de evitar *void* *. En muchos casos un casting puede significar un error de diseño.
- Evite el uso de arreglos de *char* para las cadenas. En lugar de esto, utilice la clase *string.*

Qué es C++?

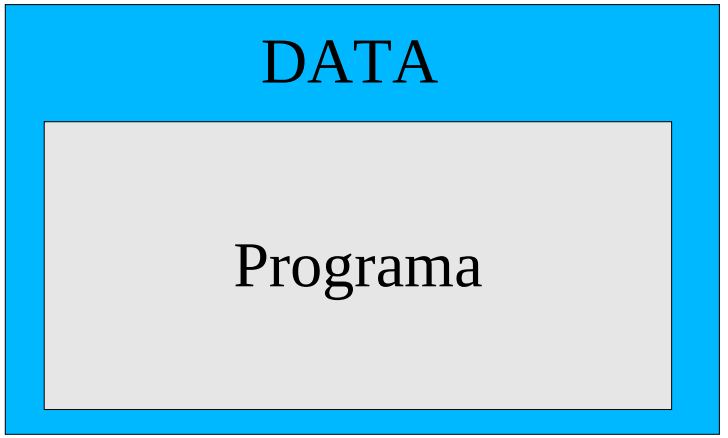
- Es un lenguaje de programación de propósito general preferiblemente para la programación sistema y que además es:
  - Un C mejorado
  - Soporta abstracción de datos
  - Soporta OxO
  - Soporta programación generica (plantillas)

Técnicas de Programación

- Programación no estructurada
- Programación procedural
- Programación modular
- Programación orientada a objetos

Programación No Estructurada

- Normalmente las personas aprenden a escribir programas, escribiendo simples algoritmos cuyas instrucciones de control operan directamente sobre la data. Es importante destacar que la data es de acceso **global.**
- La principal desventaja de este modo de programación, es que no se puede lidiar fácilmente con la escala. El programa a medida que se hace más grande se hace menos manejable.
- No se cuenta con ningún tipo de reutilización.



Programación Procedural

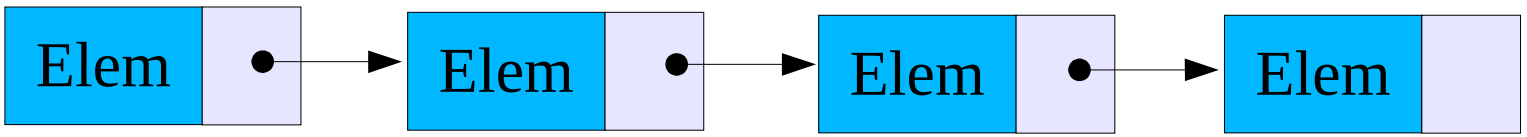
- Con la programación procedural, se obtienen trozos de código que son completamente reutilizables y se encuentran confinados en un mismo sitio. Estos trozos de código, en ocasiones llamados funciones, pueden ser llamados mediante alguna instrucción, y luego dentro de la función se puede retornar al punto de invocación mediante alguna otra instrucción que indique el retorno.
- Otras características que hacen interesante a la programación procedural son los parámetros que dan flexibilidad, los sub procedimientos, etc.
- Estas características hacen obtener un alto grado de reutilización del código y predisponen a aumentar la correctitud del programa.
- La visión del programa cambia. Pueden verse un conjunto de llamadas a procedimientos para resolver un problema.
- Hay un programa principal que coordina el computo general y se aprovecha de los procedimientos para resolver el problema.

Programación Modular

- Mediante el paradigma de la programación modular, se pueden agrupar un conjunto de procedimientos y funciones que tienen relación y así prestar un servicio reutilizable por varios programas.
- Estos conjuntos de procedimientos son denominados **módulos**.
- Un programa modular esta compuesto de diferentes módulos que interactúan mediante el llamado a procedimiento.
- Existe un programa principal que coordina las llamadas a los procedimientos.
- Cada módulo puede manejar su propia data. Esto permite a cada módulo manejar su estado interno que es modificado por llamadas a procedimientos del modulo. Sin embargo, hay un solo estado por módulo y cada módulo existe a lo más una vez en todo el programa.

Ejemplo con Estructura de Datos

- Los programas utilizan las estructuras de datos para guardar información.
- Una de las estructuras más utilizadas son las listas.
- Las listas son bloques de información concatenadas a través de un campo apuntador.



Ejemplo con Estructura de Datos

- Las listas tienen principalmente dos procedimientos que pueden ser ejecutados sobre ellas: añadir y borrar elementos.
- La semántica de este par de procedimientos depende del programador. Por ejemplo podría cumplirse la política FIFO (primero en entrar primero en salir).

Ejemplo de Implantación

- Recuerde que para implantar la lista usted debería poder:
  - Crear la lista
  - Añadir elementos
  - Quitar elementos
- Como estamos implantando un modulo debería haber un archivo que especifique la interfaz (.h) y otro la implantación (.c)

Ejemplo con E.D.

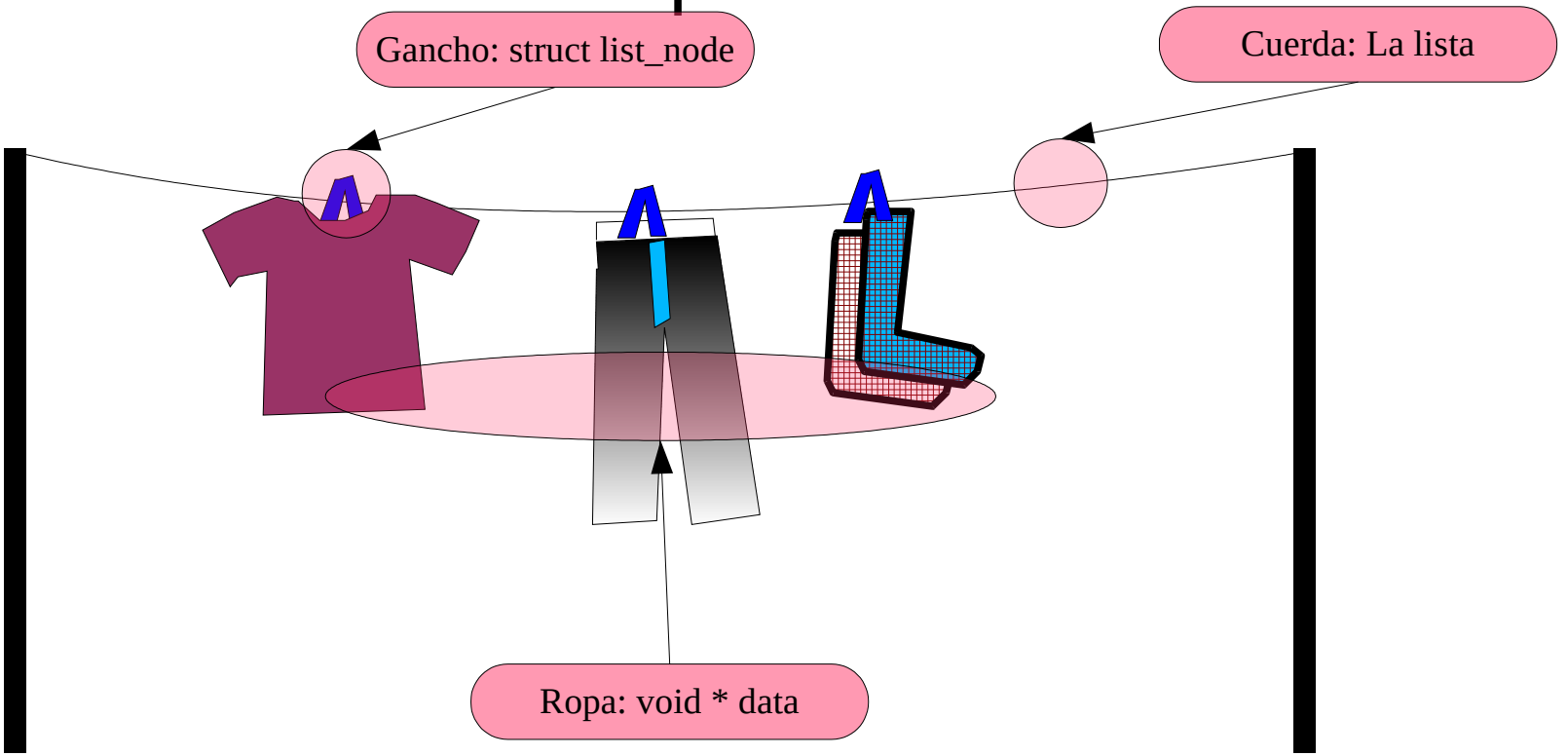
```
struct nodo_lista {
void * data;
struct nodo_lista * siguente;
} lista;

bool crear_lista();
bool anadir_elemento(void * elem);
bool borrar_elemento();  // recuerde la politica FIFO
void * obtener_el_primero();
bool esta_vacia();
```

Ejemplo con E.D.

```
bool crear_lista()
{ lista.siguiente = NULL; }
bool anadir_elemento(void * elem)
{ struct nodo_lista * nuevo;
   nuevo = (struct nodo_lista*)malloc(sizeof(struct nodo_lista));
   nuevo->data = elem;
   nuevo->siguiente = lista->siguiente;
   lista->siguiente  = nuevo; }
bool borrar_elemento();  // recuerde la politica FIFO
void * obtener_el_primero();
bool esta_vacia();
```

Analogía de una lista

- Podríamos pensar en la lista propuesta, como en un tendedero de ropa:



Ejemplo con E.D.

- La interfaz define un conjunto de “cajas negras” que dicen, que va a hacer cada operación, procedimiento, etc.
- La implantación dice como.
- La interfaz exporta las reglas de manejo del módulo.
- Algo muy importante en este ejemplo es que se dice claramente que se trabaja con “una sola” lista.

Ejemplo con E.D.

- ¿Cómo haríamos para atravesar la lista?
- ¿Qué si se requiere trabajar con más de una lista y reutilizar los procedimientos? Esto implica un cambio de escala.

```
struct nodo_lista {
void * data;
struct nodo_lista * siguente;
} lista;
// FIJESE EN EL CAMBIO DE LA INTERFAZ
struct nodo_lista * crear_lista();
bool anadir_elemento(struct nodo_lista *, void * elem);
bool borrar_elemento(struct nodo_lista *);  // recuerde la politica FIFO
```

Ejemplo con E.D.

- Observe que ahora para manejar las diferentes listas, hace falta que el procedimiento *crear_lista,* devuelva un apuntador a registro.
- En cada uno de los procedimientos especificados, se debe pasar como parámetro la dirección de la lista sobre la que se está operando.
- Observe que empezamos a generalizar la interfaz. La generalización consiste en pensar una sola vez una operación y aplicarla sobre varias instancias de una estructura.

Falta de Elegancia

- ¿Ha indagado alguna vez acerca de que operaciones de creación y transformación de la data hacen los tipos de datos básicos?
- Observe la declaración: int j;
- El entero *j* usted simplemente “lo utiliza” y no se preocupa acerca de su implantación.
- Entonces no sería mejor tener un tipo de dato lista, que se pudiera hacer:

```
lista k;
int w;
k=k+w; // añadir el elemento w a k.
```

Sobre la palabra Elegancia

- Elegante: Que muestra refinamiento o buen gusto en su aspecto. Proviene del latin *elegantem*, acusativo de *elegans* (radical *elegant*).
- Del latin, *elegantia*, significa corrección y claridad [del estilo].
- Tenga presente que el cambio de paradigma de la forma procedural a OxO nos arrojara una manera de expresarnos más **elegante**.

Debilidades de la P.M.

- Creación y destrucción explicita.
- Cada vez que se necesite una lista debe hacerse el siguiente par de operaciones:
  - Declarar el manejador: struct nodo_lista * lista;
  - Mandar a crear la lista: lista = crea_lista();
- Cuando se haya dejado de utilizar la lista debe también destruirse explícitamente. Recuerde que hay memoria dinámica de por medio.

Destrucción de la Lista

```
struct nodo_lista * c, l = crea_lista();
...
for (struct nodo_lista * n=l; n->siguiente!=NULL;
   n=c)
{    c = n->siguiente;
     free(n);   }
free(n);
```

Debilidades de la P.M.

- No hay una unión explicita entre la Data y las Operaciones.
- Esto conlleva a que la data sea definida en base a las operaciones.
- Hay que proveer explícitamente la data a las operaciones.

Debilidades de la P.M.

- No hay seguridad en los tipos de datos.
- Suponga el siguiente escenario:
  - Un dato de tipo entero: int d;
  - Un dato de tipo real: float r;
  - Una lista cualquiera llamada l.
  - Inserción de “d”: anadir_elemento(l, &d);
  - Inserción de “r”: anadir_elemento(l, &r);

Debilidades de la P.M.

- ¿Puede darsele la vuelta?
- R: Si.
- Se puede añadir información extra acerca del tipo. Pero esto recarga a la operación.

Solución a las debilidades

- La solución a las debilidades propuestas es:

  La programación Orientada a Objetos.
- Pueden tenerse la misma definición de los datos, con instancias diferentes, manejados por las mismas funciones.
- Y eso es todo?
  - R: No, por fortuna... Existen otras carácteristicas y mecanismos de la OxO que la hacen muchisimo más poderosa que la programación modular.

Programación Orientada a Objetos

- La programación orientada a objetos es una técnica cuyo bloque de construcción principal son los objetos y las interrelaciones.
- Un objeto son estructuras de datos más las funciones que operan sobre estas estructuras.

Ejercicio

- ¿Recuerda el problema del robot? ¿Cómo se modelaría según la programación modular? ¿Según la programación orientada a objetos?
- Reflexione acerca de lo que desea modelar. Cuales son los entes que intervienen.
- Recuerde el repaso: estructuras, punteros, matrices.