LENGUAJE C++

Tema 1 – Introducción
Elementos del Lenguaje

Abstracción

- [Wulft]:La Humanidad ha desarrollado una técnica para tratar la complejidad: La Abstracción.
  - *Permite representar las características esenciales de un objeto real, sin preocuparse de sus restantes características.*
  - Es la capacidad para encapsular y aislar información, del diseño y ejecución.
  - Es Esencial para el funcionamiento de la mente humana normal.
- Una Abstracción (Modelo Mental) debe ser mas sencillo que el sistema al cual imita.
  - El mapa como modelo de la carretera: indica su topografía, altura, distancias, etc, mientras ignora detalles irrelevantes (flores, material de la carretera, etc.)
- La Historia inicia con los Procedimientos, continúa con los Módulos, pasa por el TAD y termina con los Objetos.

Programación Estructurada

- Técnica en la cual una solución se modela a partir de 3 estructuras lógicas de control:
  - ***Secuencia:*** Sucesión simple de dos o mas Operaciones.
  - ***Selección (if)***: bifurcación condicional de una o mas operaciones.
  - ***Iteración (while, for):*** Repetición de una operación mientras se cumple una condición.
- Un programa estructurado está compuesto de segmentos. C/U de ellos:
  - Tiene solamente **una entrada y una salida**,
  - Están compuestos por combinaciones arbitrarias de las estructuras lógicas de control.
- Los Programas pueden ser leídos en secuencia, desde el comienzo hasta el final sin perder la continuidad de la tarea del programa
  - Es una consecuencia de utilizar las 3 estructuras y de eliminar la instrucción de desvío de flujo de control.

Programación Orientada a Objetos

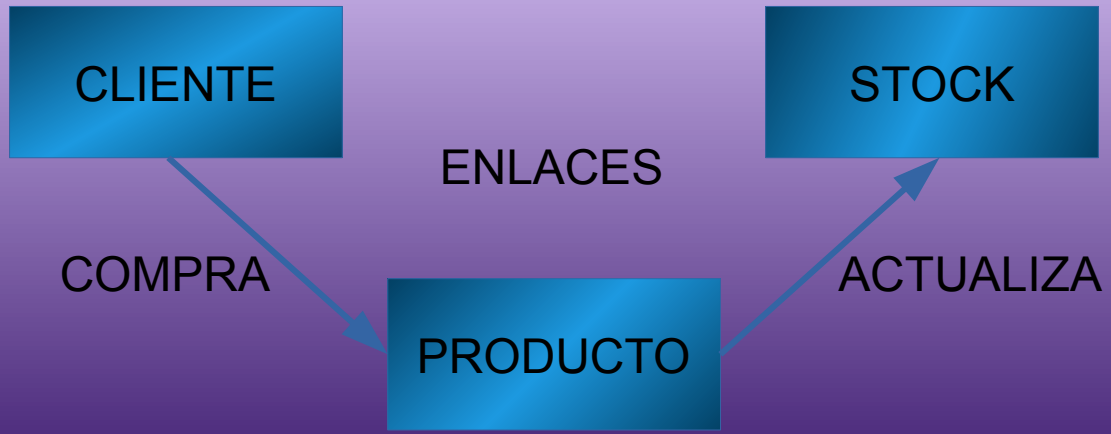
- Facilita la construcción de sistemas complejos a partir de componentes llamados Objetos.
  - Tiene los aciertos de la Programación Estructurada agregándole y combinándole con el OBJETO.
- **El "OBJETO"**:
  - Abstrae un "objeto" de la realidad tan fielmente como sea necesario.
  - Es una "Entidad" percibida dentro del sistema.
  - Encapsula Métodos y Propiedades.
    - Métodos: pueden responder a la Programación Estructurada.

Las 7 innovaciones del OBJETO

- **Clases:** define un “tipo” de Objeto.
- **Propiedades:** Datos propios de un Objeto.
- **Métodos:** Forma de manipular los Objetos. Funciones.
- **Instancias:** Una copia de la Zona de Datos de un Objeto.
- **Mensajes:** Usados para comunicar Objetos. Se implementan mediante *Métodos*.
- **Herencia:** permite a diferentes Objetos compartir código común. Incrementa la funcionalidad.
- **Poliformismo:** Es la propiedad mediante un Objeto puede tomar muchas formas: Un mismo método en objetos diferentes tendría comportamientos diferentes.

Características de un Objeto

- ***Estado de un Objeto:*** conjunto de valores de las propiedades en un momento dado.
- ***COMPORTAMIENTO:*** Aquello que el objeto puede hacer.
- ***IDENTIDAD:*** conjunto de características que permiten diferenciar a un objeto de los demás.
- ***Enlace:*** Conexión física o conceptual entre objetos.



Programación Orientada a Objetos Vs. Programación Estructurada

	PE	*POO*
Un Programa está compuesto por	Conjunto de Módulos y/o procedimientos	Conjunto de objetos que se comunican entre si con un objetivo común.
Longitud de código	Mayor a POO en 40%	Menor a la PE en 40%
Reutilización de SW	Sino fueron creados como TAD, se hace difícil y poco confiable	Confiables y fáciles de montar
Forma de Reutilizar	Biblioteca de funciones y librerías	Catálogos y componentes
Modela centrándose en	Operaciones	Los Objetos de la realidad y su comunicación
Idea Clave	Descomponer una Aplicación en programas más pequeños	Reflejar el mundo real mediante el ensamblado de clases

Ventajas y desventajas de POO

- Ventajas
  - Objetos Bien Diseñados e Implementados:
    - Absolutamente independientes del contexto.
    - Alta mantenibilidad
  - Mayor Reutilización: biblioteca de clases bases que serán personalizadas en futuros desarrollos.
  - Códigos mas pequeños.
- Desventajas
  - Gran esfuerzo de Análisis y Diseño de objetos.
  - La depuración de POO es más compleja que la PE.
    - Si se produce un error hay que recorrer el árbol de herencia.

Evolución de C a C++

- ’70: Se diseña el Lenguaje C.
  - Inventado y desarrollado por Dennis Ritche y Ken Thompson
  - Es el resultado de un proceso de desarrollo comenzado con el lenguaje B
    - Al B le resultaba complejo manejar programas de mas de 100.000 líneas.
- ’80: Bjarne Stroustrup agrega extensiones de *Clases* al Lenguaje C y crea el Lenguaje ***C++***

El lenguaje C++

- Al diseñar C++ Stroustrup alcanzó los Objetivos de
  - *C++ mantiene la eficiencia del C*
  - *C++ es un Lenguaje de nivel medio*
  - *El programador sigue al Mando.*
  - *La Potencia de los Objetos.*
- C++ es utilizable para cualquier problema computacional.
  - Editores (Geany), bases de datos (Oracle), sistemas de archivos (Linux), programas de comunicaciones.
- Los Programadores C pueden empezar inmediatamente con las Extensiones C++.
  - Casi todo lo que se conoce de C, es aplicable a C++.
  - PERO: para escribir código eficiente OO, hay que cambiar el Paradigma.

Un primer programa en C++

```
#include <iostream>
int main (int argc, char *argv[])
{
    std::cout<<"Hola Matias"<<std::endl;
    return 0;
}

------El mismo pero agregando namespace------
#include <iostream>
using namespace std;

int main (int argc, char *argv[])
{
    cout<<"Hola Matias"<<endl;
    return 0;
}
```

La instrucción **`using namespace`** especifica que los miembros de un namespace van a utilizarse frecuentemente en un programa.
Esto permite al programador tener acceso a todos los miembros del namespace y escribir instrucciones mas concisas.

Elementos básicos de un programa

- ***Comentarios:***
  - // comentario de una sola línea.
  - /* comentario de varias líneas. */
- ***Bloques de código:***
  - { Todos los bloques de código van entre llaves }
- ***Fin de línea:***
  - ; Todas las instrucciones finalizan con un punto y coma

```
#include <iostream>
using namespace std;
/* primer programa
Muestra por pantalla 10 veces
"Hola Matias" */
int main()
{
    int i;
    for (i=0; i<10; i++)
    {
        cout << "Hola Matias" <<
  endl;
    } //fin for
    return 0;
} //fin main
```

Tipos de datos

- **Tipos primitivos:** incluidos en el compilador
  - `short, int, long, long long,`
  - `char,`
  - `float, double, long double,`
  - `void`

Se extienden con **`signed`** y **`unsigned`**

- **Tipos derivados**: creados a partir de tipos primitivos
  - Estructuras
  - Uniones
  - Vectores y Matrices
  - Punteros
  - Clases

Palabras reservadas o claves

- El primero contiene las palabras de C y que C++ como evolución de C también contiene:

```
auto      const     double    float     int     short     struct    unsigned
break     continue  return    for       long    signed    switch    void
case      default   enum      goto      do      sizeof    typedef   volatile
char      register  extern    if        else    static    union     while
```

- Palabras que no provienen de C y que, por tanto, solo utiliza C++:

```
asm       dynamic_cast  namespace     reinterpret_cast    try  bool
explicit  new           static_cast   typeid              catch  false
operator  template      typename      class               friend  private
this      using         const_cast    inline              public  throw
virtual   delete        mutable       protected           true    wchar_t
```

- Las siguientes palabras reservadas se han añadido como alternativas para algunos operadores de C++ y hacen los programas más legibles y fáciles de escribir:

```
and  bitand   compl    not_eq   or_eq    xor_eq   and_eq  bitor
not  or       xor
```

Operadores Aritméticos

Operador	Nombre	Descripción
+	Suma	5+2 → 7
-	Resta	5-2 → 3
*	Multiplicación	5*2 → 10
/	División	5/2 → 2
%	Módulo	5%2 → 1
(tipo de dato)	“Cast” forzado	(double)5 → 5.0

Operadores Relacionales

Operador	Nombre	Descripción
==	Igual a	if (a==‘s’)
!=	Diferente de	if (a!=null)
>	Mayor que	if (a>0.5)
<	Menor que	if (a<2l)
>=	Mayor o igual que	if (a>=2f)
<=	Menor o igual que	if (a<=3)

Operadores Lógicos

Operador	Nombre	Descripción
&&	Y (AND)	if ((a>3) && (a<9))
||	O (OR)	if ((a==2) || (a==3))
!	NEGADO (NOT)	if (!(a==3)) es igual a if (a!=3)

Importante:

FALSO es igual a cero y se representa con false o 0.

VERDADERO es igual a uno y se representa con true o 1.

Operadores de Asignación

Operador	Abreviado	No Abreviado
=	a=2;	a=2;
++	n++; o ++n;	n=n+1;
--	n--; o --n;	n=n-1;
+=	n+=2;	n=n+2;
-=	n-=2;	n=n-2;
*=	n*=2;	n=n*2;
/=	n/=2;	n=n/2;
%=	n%=2;	n=n%2;

Operadores de Bits

Operador	Nombre	Descripción
<<	Corrimiento a la izquierda	b=a>>2;
>>	Corrimiento a la derecha	b=a<<3;
&	Y (AND) entre bits	c=a&128;
|	O (OR) entre bits	c=a|0x0a;
~	Complemento A1	c=~a;
^	O exclusivo (XOR)	c=^a;

Precedencia de operadores

( ) [ ] ->	Alta prioridad
! ~ + - ++ -- & * sizeof	Unarios
* / % + -	Aritméticos
<< >>	Corrimiento de bits
< <= > >= == !=	Relacionales
& ^ | && || ?:	Bits / Lógicos / Condicional
= *= /= %= += -= &= ^= |= <<= >>=	Asignación
,	Evaluación

Directivas de preprocesado

- **#include:** incluye un archivo de cabecera.
  - `#include "archivo.h"`
  - `#include <iostream>`
- **#define**: define bloques de código para su reemplazo
  - `#define TRUE 1`
  - `#define TAM 32 //constante`
  - `#define AREA(X) x*x //función`

Variables

Una variable es una **posición de memoria** cuyo valor puede ser **cambiado** durante la ejecución del programa.
Inicialmente el valor de una variable es indeterminado.
Todas las variables deben de ser declaradas para ser utilizadas.

```
<tipo de dato> <identificador>;
<tipo de dato> <identificador> = <valor>;
```

```
int nro;
float area, radio,
   volumen;
char letra = 'a';
double iva = 0,21;
```

Las variables pueden ser declaradas:

- Fuera de las funciones: Variables globales
- Dentro de una función: Variable local
- En la cabecera de una función: parámetros formales de la función (es una variable local)

Modificadores de variables

- **const:** permite asignar a una variable un valor constante, es decir que una vez asignado a dicha variable su valor no podrá ser modificado durante el programa.
  - `const <tipo dato> <identificador> = valor;`
- **Casting**: La conversión de una expresión de un tipo de dato determinado en otro tipo se conoce como conversión de tipos (type casting).

```
Conversión implícita
short a = 2000;
int b;
b = a;
```

```
Conversión explicita
float a=20,50;
int b;
b = (int) a;
b = int (a); //otra forma
```

Sentencia de selección if - else

```
if (expresión)
    sentencia;
else
    sentencia;
```

Nota: una expresión en C++ es todo aquello que regresa un valor. Como por ejemplo una condición lógica, operaciones aritméticas, llamadas a funciones, una variable, una constante (numérica, carácter, etc.).

```
if (expresión)
{
    sentencia1;
    sentencia2;
}else{
  if (expresión2)
      sentencia;
  else
      sentencia;
}
```

Sentencia de selección switch - case

```
switch(expresión)
{
    case 1:  sentencias;
                 break;
    case 2:  sentencias;
                 break;
     :
    case n:  sentencias;
                  break;
    default: sentencias_default;
                  break;
}
```

Sentencias de repetición while y do - while

WHILE

```
while (expresión)
    sentencia;

while (expresión){
   sentencia1;
   …
   sentencian;
}
```

Do - WHILE

```
do
    sentencia;
while(expresión);

do{
   sentencia1;
   …
   sentencian;
}while(expresión);
```

Sentencia de repetición for

```
for (inicialización; condición; incremento)
   acción;

for (inicialización; condición; incremento){
   acciones;
}

for(expr1;expr2;expr3)
   Sentencia;
```

- expr1 y expr 3 son asignaciones o llamadas a funciones.
- expr2 es una expresión relacional.

- Si expr2 no esta presente se considera siempre verdadero →bucle infinito, unica salida break o return.

Sentencias de salto break y continue

- **break:**
  - Permite controlar las salidas de los bucles.
  - Provee una salida temprana para for, while, do, switch.
  -

```
for(i=0;i<1000;i++){
    //hacer algo
    if(kbhit()) break;
}
```

- **continue**:
  - Es utilizada cuando la parte del bucle que sigue es complicada.
  - Provoca la próxima iteración del bucle cerrado a ser ejecutado inmediatamente.

Bibliografía & Licencia

- *Como programar en C++, 9na Ed*, Deitel, H.M. y Deitel, P.J., Pearson
- *Programación en C++, Un enfoque práctico*, Joyanes Aguilar, L., McGraw-Hill.
- *Thinking in C++, 2da Ed*, Bruce Eckel, Prentice Hall PTR.

- Autor:

Matías E. García

Prof. & Tec. en Informática Aplicada

www.profmatiasgarcia.com.ar

info@profmatiasgarcia.com.ar