



# Unidad VII Conceptos avanzados

M.C. Juan Carlos Olivares Rojas



# Agenda

- 7.1 Almacenamiento dinámico
- 7.2 Entrada y salida con formato
- 7.3 Manejo de archivos
- 7.4 Funciones en línea
- 7.5 Enlace con lenguaje ensamblador
- 7.6 Argumentos en línea
- 7.7 Empleo de polimorfismo
- 7.8 Manejo de excepciones

# 7.1 Almacenamiento dinámico

- La declaración de variables se realiza de manera automática de forma estática es decir una vez declarada una variable su espacio se ocupa durante toda la ejecución de un programa.
- Con el almacenamiento dinámico se puede utilizar más memoria de lo normal y ese tamaño puede crecer o decrecer según se necesite

# Almacenamiento dinámico

- La memoria dinámica se utiliza para implementar estructuras de datos complejas y variables como listas, árboles, colas, grafos, etc.
- En C, la memoria dinámica se crea a través de las funciones `malloc()` utilizando de por medio punteros y se libera con la función `free`

# Almacenamiento dinámico

- En C++ se utiliza los operadores `new` y `delete` para las mismas operaciones;
- `int *a;`
- `A = new int[1000];`
- `delete A;`

## 7.2 Entrada y salida con formato

- La entrada en C se realiza a través de la función `scanf()` y la salida con `printf()`. Se utilizan algunos parámetros opcionales como delimitadores para los tipos y tamaño de los datos pero aún así son independientes.
- C++ propone un flujo de entrada/salida independiente del tipo de datos a través de las funciones `cin` y `cout`;

# Entrada y salida con formato

- Para utilizar el flujo de datos de C++ se debe hacer uso de la biblioteca estándar `iostream.h`
- `cout` se utiliza para la salida estándar, `cin` para la entrada estándar y `cerr` para el error estándar.
- Se utilizan los operadores `<<` para redireccionar el flujo de salida y `>>` para redireccionar el flujo de entrada.

# Entrada y salida con formato

- La entrada estándar al igual que en C está asociado al teclado, la salida y el error estándar a la pantalla.
- Los flujos pueden cambiarse.
- Para imprimir algún dato:  
`cout<<"Hola"<<a<<b<<endl` donde a y b son enteros

# Entrada y salida estándar

- Existen algunos operadores especiales como endl para el salto de una línea.
- La entrada se realiza de la siguiente forma  
cin>>cadena>>entero>>flotante
- La entrada y salida se simplifica haciendolo transparente para el programador

## 7.3 Manejo de archivos

- Los archivos en C se manejan utilizando la estructura FILE definida en la librería estándar. La operación de un archivo puede ser a través de modo texto o binario.
- FILE \*p;
- p =fopen(“archivo.txt”,”r”);
- fwrite(p, dato, tamaño, veces);
- fread();

# Manejo de archivos

- Todo en un sistema operativo puede verse como un archivo, los datos, los dispositivos físicos, los procesos, etc.
- Existen operaciones en modo crudo como write, read, open, etc.
- También existen funciones de entrada y salida de texto como fprintf, fscanf, etc.

# Escribir archivo

```
#include <fstream.h>
int main()
{
ofstream archivo; // objeto de la clase ofstream
    archivo.open("datos.txt");
archivo << "Primera línea de texto" << endl; archivo
    << "Segunda línea de texto" << endl; archivo <<
    "Última línea de texto" << endl; archivo.close();
return 0;
}
```

# Leer archivo

```
#include <fstream.h>
int main()
{
ifstream archivo("Besos.txt", ios::noreplace);
char linea[128];
long contador = 0L;
if(archivo.fail())
    cerr << "Error al abrir el archivo Besos.txt" << endl;
else
    while(!archivo.eof()) {
        archivo.getline(linea, sizeof(linea));
        cout << linea << endl;
        if(++contador % 24==0) {
            cout << "CONTINUA..."; cin.get();
        }
    }
    archivo.close();
return 0;
}
```

## 7.4 Funciones en línea

- Las funciones inline indican al compilador que se copie la funcionalidad del método entero repitiendo código para mejorar el rendimiento.
- Las funciones en línea son recomendaciones al igual que las variables registro (utiliza los registros del procesador para almacenar valores) por lo que el compilador se reserva el derecho de hacerlo

# Funciones en línea

- `inline int clase::método(int a, int b);`
- `register int a;`
- En lugar de gastar tiempo haciendo saltos de función, se accede directamente, una especie de macros. Las funciones inline de gran tamaño no se optimizan.

## 7.5 Enlace con lenguaje ensamblador

- Es posible agregar instrucciones de lenguaje ensamblador en C/C++ para aumentar la eficiencia del código generado.
- La inclusión de lenguaje ensamblador así como otros lenguajes embebidos dependerá del compilador.
- La gran mayoría de compiladores de C/C++ soporta el lenguaje ensamblador.

# Enlace con lenguaje ensamblador

- El siguiente código suma dos números en ensamblador para intel x86.

```
asm
{
  mov ax, 10;
  add ax, 25;
}
```

## 7.6 Argumentos en línea

- Los argumentos en línea nos sirven para obtener datos de manera directa desde el programa.
- Por ejemplo al hacer una suma ocupamos dos valores, los cuales leemos a partir del usuario por algún tipo de interfaz.
- Se pueden pasar argumentos desde la línea de comandos de la siguiente forma

# Argumentos de línea

- Suma 10 12
- El primer argumento es el nombre del programa y esta en la posición 0.
- Los argumentos se leen a través de la función `main(int argc, char *argv[])`. En donde `argc` contiene el número de argumentos y `argv` contiene los argumentos en forma de cadena.

## 7.7 Empleo de polimorfismo

- El polimorfismo no sólo se emplea para la sobrecarga de funciones y operadores. También se emplea en la creación de clases que pueden cambiar en tiempo de ejecución.
- Para este tipo de actividades se tiene que definir una función como abstracta.

# Ejemplos de polimorfismo

- Un objeto puede tener muchas formas dependiendo de su estado. Por ejemplo, el agua puede ser sólida si la temperatura es menor a 0 grados centígrados, vapor si es mayor de 100 grados centígrados o líquida en estado normal.
- Una clase abstracta es aquella de la cual no se crean instancias del objeto.

# Ejemplo de polimorfismo

- Las clases abstractas en C++ deben tener algún método definido como virtual.
- El atributo virtual indica que ese método será implementado en una clase que hereda de la clase abstracta, por lo que dicha clase recibe el nombre de interfaz.
- Los métodos se ligan de manera automática en tiempo de ejecución.

## 7.8 Manejo de excepciones

- Las excepciones son una manera elegante de manejar errores.
- Todo bloque de instrucciones entre try {} es verificado en busca de errores.
- Si se produce un error estos son capturados en el bloque instrucción catch(excepcion i) {}

# Manejo de excepciones

- Algunos ejemplos básicos de excepciones incluyen división entre 0, índice de arreglos fuera de rango, memoria dinámica, etc.
- Es una mejora a utilizar tipos de datos enteros para mandar errores.

# ¿Preguntas?

