Resumen general

0

Temas a desarrollar

El contenido total del documento abarca:

1. Introducción a la algoritmia
2. Introducción a C, C++
3. Tipos de datos
    a. Datos simples
    b. Estructuras de datos
4. Asignación, análisis de caso repeticiones
5. Patrones algorítmicos simples
6. Struct
7. Array
8. Flujos
9. Estructuras enlazadas
10. Combinaciones de estructuras
11. Patrones de carga, recorrido, búsqueda y ordenamiento
12. Criterios de selección de estructuras de datos y de control

Introduccion
Introducción a la algoritmia
Entorno de desarrollo C-C++
Tipos de datos
Estructuras de control
Asignación Análisis de caso Ciclos
Patrones algorítmicos simples
Intercambios
Máximos y mínimos: distintos lotes, distintos criterios
Seguidillas
Ejercicios integradores
Práctica

Funciones

Tema
Definiciones y declaraciones
Declaración, definición, prototipos, invocación, bibliotecas
Intercambio de información
Argumentos, parámetros: por valor, por referencia
Reusabilidad – Generalidad
Concepto de reusabilidad
Concepto de generalidad
Ejercicios integradores
Práctica

Struct

Tema
Definiciones y declaraciones
Combinación de estructuras: struct con un campo struct
Asignación interna y externa

Array – Flujos – Estructuras enlazadas

Tema
Necesidad de su uso. Definiciones y declaraciones
Vectores y matices – Flujos – Estructuras enlazadas
Datos simples, estructuras
Combinación de estructuras
Recorridos
Secuencial
En un rango
Con Corte de control (criterio único/criterio múltiple)
Apareo (criterio único/criterio múltiple)
Búsqueda
Directa
Secuencial
Dicotómica (criterio único/criterio múltiple)
Carga
Directa
Secuencial
Ordenada
Ordenamiento
Con Posición Única Predecible
Método de ordenamiento
Ejercitación
Ejercicios integradores

Programación

La programación es una actividad transversal asociada a cualquier área de la informática, aunque es la ingeniería del software el área específica que se ocupa de la creación del software.
En principio la programación se veía como un arte, solo era cuestión de dominar un lenguaje de programación, esto fue ampliamente superado

Programa:

Programa: conjunto de instrucciones no activas almacenadas en un computador, se vuelve **tarea** a partir de que se selecciona para su ejecución y permite cumplir una función específica. Un **proceso** es un programa en ejecución.

Dato

Dato representación de un objeto el mundo real mediante el cual se pueden modelizar aspectos de un problema que se desea resolver con un programa en una computadora.
<dato> -> <objeto><atributo><valor> → semáforo, luz, roja

Información

Interpretar el dato: semáforo, luz, roja → supone un alerta

Conocimiento

Que hacemos con esa información, que acciones son las apropiadas realizar en función de la información disponible y el propósito o fin a resolver, el conocimiento nos permite abordar soluciones que no necesariamente son únicas y muchas veces dependen del contexto
Semaforo, luz, roja:
→ peligro me detengo
→miro hacia ambos lados y veo de cruzar con cuidado
→en calle de doble mano miro primero a la izquierda y luego a la derecha o primero a la derecha y luego a la izquierda dependiendo del sentido de circulación

Lenguaje de programación

Conjunto de instrucciones permitidas y definidas por sus reglas sintácticas y su valor semántico para la expresión de soluciones de problemas.

Tecnologías/Frameworks FrontEnd (Web)	Lenguajes Backend
HTML (paginas web)	Javascript (*)
CSS (ESTILOS)	Python(PROPOSIVO GRAL LEGIBILIDAD INTERP)
Javascript (*)(AGREGA INTERACCION)	PHP(INDEPENDIENTE DE LA PLATAFORMA)
React(BIBLIOTECA JS)	Ruby(GRAL OO INTER.)
Redux(BIBLIOTECA INTERFACE USUARIO)	C C++ C# JavaVB
Angular(CREAR Y MANTENER PAGINAS WEB)	Go(MULTIPLAT. NO 100% OBJETO NO TD GENERICO)
Bootstrap(aplicaciones web)	

Algoritmo

Algoritmo
Especificación rigurosa (debe expresarse en forma univoca) de la secuencia de pasos, instrucciones, a realizar sobre un autómata para alcanzar un resultado deseado en un tiempo finito. Esto último supone que el algoritmo empieza y termina, en el caso de los que no son de tiempo finito (ej. Sistemas en tiempo real) deben ser de número finito de instrucciones.

Un algoritmo debe tener al menos las siguientes características:

1. **Ser preciso**:
2. **Ser definido**.
3. **Ser finito**:
4. **Presentación formal**:
5. **Corrección**:
6. **Eficiencia**:

Propiedades de los algoritmos

1. Especificación precisa de la entrada:
2. Especificación precisa de cada instrucción:
3. Un algoritmo debe ser exacto y correcto, tener etapas bien definidas y concretas.
4. Debe ser fácil de entender, codificar y depurar.
5. Debe hacer uso eficiente de los recursos de la computadora

Identificadores

Nombre simbolico que define quien programa para denotar "identificar" ciertos elementos en la aplicación o programa.
Estos elementos pueden ser:

1. Variables
2. Constante
3. Funciones

ValorL

Declaraciones y definiciones

Una Declaracion es una construcción que especifica las propiedades de un identificador
Si una declaración de una constante o variable especifica el valor permanente de la constante o el valor inicial de una variable, algunos lenguajes lo llaman definición

Expresiones y sentencias

Expresión conjunto de operadores y operandos que reducen a un valor
Una sentencia es una acción que se ejecuta efectivamente. Pueden ser:

1. Simples → una única acción → asignación
    a. Interna
    b. Externa
        i. Entrada
        ii. salida
2. Estructurada → responden a un formato o estructura determinada → análisis de caso, repeticiones.

a. Análisis de caso
    i. Simple
        1. Completo
        2. Incompleto
    ii. Compuesto
        1. Completo
        2. Incompleto
b. Iteraciones
    i. Exactas
    ii. No excactas
        1. Precondicionales
        2. Poscondicionales

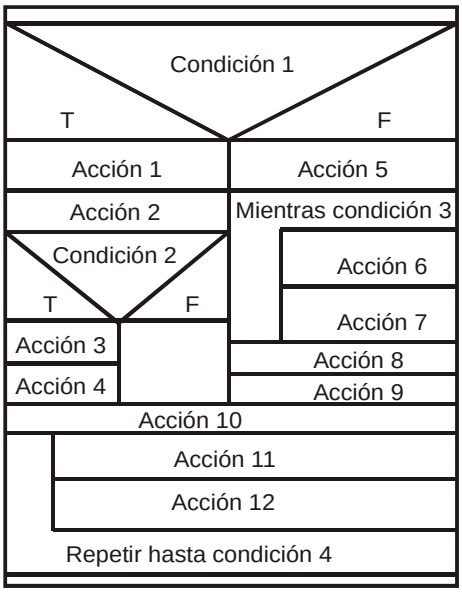
3. Compuestas→ una o mas sentencias simples, estructuradas ocombinaciones que se tratan como una unidad **{**sentencia;… ;sentencia**}**

Solucion en un LP (C++)

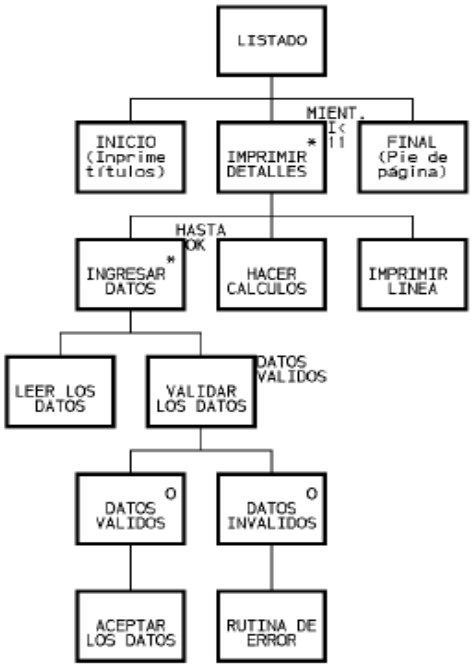
```
#include <iostream>//pone a disposición elementos que necesita
using namespace STD; //orienta en la búsqueda de dispositivos de E/S
int main() {
        char nombre[10]//declara donde guardar el nombre en la memoria
        cout<<"Ingrese un nombre: ";//deriva un  mensaje a la pantalla
        cin>>nombre;//recibe un valor desde el teclado y lo almacena en el espacio reservadp
        cout<< "el nombre ingresado es : " <<nombre;//deriva mensaje e identificsdor
        return 0;// termina el prorama
}
```

Representaciones graficas para la formalización y resolución estratégica de problemas computables de información

Diagrama de Nassi-Sneiderman

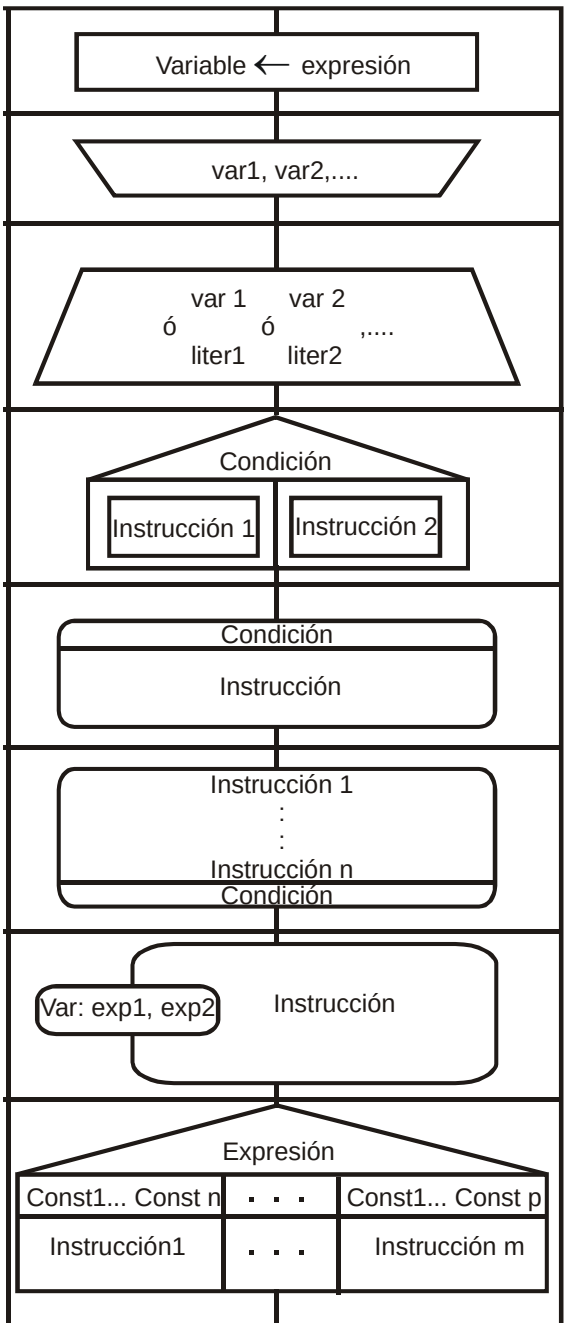


Diagramas de Jackson

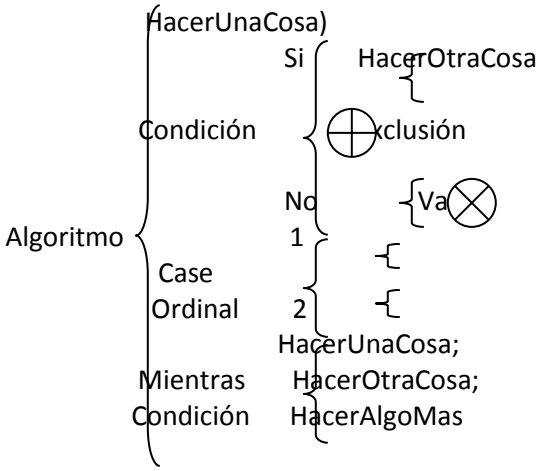


Diagramas de Lindsay.

DIAGRAMA DE DETALLE



Llaves de Warniel



Equivalencias entre notación algorítmica y lenguajes de programación

CONDICIONAL

Formato	C
SI Condicion **ENTONCES** **S** **SI_NO** **R** **FIN_SI**	**If (expresion)** **S;** **else** **R;**

Formato	C
SEGÚN expr **V1 : S1** **V2 : S2** **EN_OTRO_CASO : Sn** **FIN_SEGÚN**	**switch (selector) {** **case etiqueta:S; break;** **.....** **default: R;** **}**

ITERACION

Formato	C
Mientras Cond. Hacer **S** **FIN_MIENTRAS**	**while(expresion)** **S;**

Formato	
REPETIR **S** **MIENTRA Cond**	**do** **S;** **while(expresion)**

Formto	C
PARA i [Vi..Vf] HACER **S** **FIN_PARA**	**for(i=0;i<vf;i++)** **S;**

Estilos de Indentación

Recomendacion de estilos de indentacion para desarrollos más claros y legibles.(Ing. J. M. Sola)

Estilo The One True Brace Style

```
while( SeaVerdad() ) {
      HacerUnaCosa();
      HacerOtraCosa();
}
HacerUnaUltimaCosaMas();
```

BSD/Allman.

```
while( SeaVerdad() )
{
      HacerUnaCosa();
      HacerOtraCosa();
}
HacerUnaUltimaCosaMas();
```

Definiciones

Abstracción

Proceso de análisis del mundo real con el propósito de interpretar los aspectos esenciales de un problema y expresarlo en términos precisos.

Modelizacion

Abstraer un problema del mundo real y simplificar su expresión, tratando de encontrar los aspectos principales que se pueden resolver, requerimientos, los datos que se han de procesar y el contexto del problema.

Precondición

Información conocida como verdadera antes de iniciar el programa.

Poscondición

Información que debiera ser verdadera al cumplir un programa, si se cumple adecuadamente el requerimiento pedido.

Especificación

Proceso de analizar problemas del mundo real y determinar en forma clara y concreta el objetivo que se desea. Especificar un problema significa establecer en forma univoca el contexto, las precondiciones el resultado esperado, del cual se derivan las poscondiciones.

Tipos de datos

Los tipos de datos Identifican o determinan un dominio de valores y un conjunto de operaciones aplicables sobre esos valores.

1. Primitivos.
2. Derivados.
3. Abstractos.

Tipos de datos pueden ser.

1. **Estáticos:** Ocupan una posición de memoria en el momento de la definición, no la liberan durante el proceso solamente la liberan al finalizar la aplicación.
    a. **Simples**: Son indivisibles en datos mas elementales, ocupan una única posición para un único dato de un único tipo por vez.
        i. **Ordinales**: Un tipo de dato es ordinal o esta ordenado discretamente si cada elemento que es parte del tipo tiene un único elemento anterior (salvo el primero) y un único elemento siguiente (salvo el ultimo).
            1. **Enteros**: Es el tipo de dato numérico mas simple.
            2. **Lógico** o booleano: puede tomar valores entre dos posibles: verdadero o falso.
            3. **Carácter**: Proporcionan objetos de la clase de datos que contienen un solo elemento como valor. Este conjunto de elementos esta establecido y normatizado por el estándar ASCII.
        ii. **No ordinales**: No están ordenados discretamente, la implementación es por aproximación
            1. Reales: Es una clase de dato numérico que permite representar números decimales.
    b. **Cadenas**: Contienen N caracteres tratados como una única variable.
    c. **Estructuras**: Tienen un único nombre para mas de un dato que puede ser del mismo tipo o de tipo distinto. Permiten acceso a cada dato particular y son divisibles en datos mas elementales.
    Una estructura es, en definitiva, un conjunto de variables no necesariamente del mismo tipo relacionadas entre si de diversas formas.
    Si los datos que la componen son todas del mismo tipo son homogéneas, heterogéneas en caso contrario.
    Una estructura es estática si la cantidad de elementos que contiene es fija, es decir no cambia durante la ejecución del programa
        i. **Registro**: Es un conjunto de valores que tiene las siguientes características:
        Los valores pueden ser de tipo distinto. Es una estructura heterogénea.
        Los valores almacenados se llaman campos, cada uno de ellos tiene un identificador y pueden ser accedidos individualmente.
        El operador de acceso a cada miembro de un registro es l operador punto**.**
        El almacenamiento es fijo.
        ii. **Arreglo**: Colección ordenada e indexada de elementos con las siguientes características:
        Todos los elementos son del mismo tipo, un arreglo es una estructura homogénea.
        Los elementos pueden recuperarse en cualquier orden, simplemente indicando la posición que ocupa dentro de la estructura, esto indica que el arreglo es una estructura indexada.

El operador de acceso es el operador []
La memoria ocupada a lo largo de la ejecución del programa es fija, por esto es una estructura estática.
El nombre del arreglo se socia a un área de memoria fija y consecutiva del tamaño especificado en la declaración.
El índice debe ser de tipo ordinal. El valor del índice puede verse como el desplazamiento respecto de la posición inicial del arreglo.
Los arreglos pueden ser de varias dimensiones. Esta dimensión indica la cantidad de índices necesarias para acceder a un elemento del arreglo.
El arreglo lineal, con un índice, o una dimensión se llama vector.
El arreglo con 2 o mas índices o dimensiones es una matriz. Un grupo de elementos homogéneo con un orden interno en el que se necesitan 2 o mas índices para referenciar a un elemento de la estructura.

iii. **Archivos**: Estructura de datos con almacenamiento físico en memoria secundaria o disco.
Las acciones generales vinculadas con archivos son
Asignar, abrir, crear, cerrar, leer, grabar, Cantidad de elementos, Posición del puntero, Acceder a una posición determinada, marca de final del archivo, definiciones y declaraciones de variables.
Según su organización pueden ser secuenciales, indexados.

1. **Archivos de texto**: Secuencia de líneas compuestas por cero uno o mas caracteres que finalizan con un carácter especial que indica el final de la línea. Los datos internos son representados en caracteres, son mas portables y en general mas extensos.
2. **Archivos de tipo o binarios**: secuencia de bytes en su representación interna sin interpretar. Son reconocidos como iguales si son leídos de la forma en que fueron escritos. Son menos portables y menos extensos.

2. **Dinámicos:** Ocupan direcciones de memoria en tiempo de ejecución y se instancian a través de punteros. Esta s instancias pueden también liberarse en tiempo de ejecución. El tema de puntadores y estructuras enlazadas (estructuras relacionadas con este tipo de dato se analizan en detalle en capítulos siguentes)
   a. **Listas simplemente enlazadas**: cada elemento sólo dispone de un puntero, que apuntará al siguiente elemento de la lista o valdrá NULL si es el último elemento.
   b. Pilas: son un tipo especial de lista, conocidas como listas LIFO (Last In, First Out: el último en entrar es el primero en salir). Los elementos se "amontonan" o apilan, de modo que sólo el elemento que está encima de la pila puede ser leído, y sólo pueden añadirse elementos encima de la pila**.**
   c. **Colas**: otro tipo de listas, conocidas como listas FIFO (First In, First Out: El primero en entrar es el primero en salir). Los elementos se almacenan en fila, pero sólo pueden añadirse por un extremo y leerse por el otro.
   d. **Listas circulares**: o listas cerradas, son parecidas a las listas abiertas, pero el último elemento apunta al primero. De hecho, en las listas circulares no puede hablarse de "primero" ni de "último". Cualquier nodo puede ser el nodo de entrada y salida.
   e. **Listas doblemente enlazad**as: cada elemento dispone de dos punteros, uno a punta al siguiente elemento y el otro al elemento anterior. Al contrario que las listas abiertas anteriores, estas listas pueden recorrerse en los dos sentidos.

f. **Árboles**: cada elemento dispone de dos o más punteros, pero las referencias nunca son a elementos anteriores, de modo que la estructura se ramifica y crece igual que un árbol.
g. **Árboles binarios**: son árboles donde cada nodo sólo puede apuntar a dos nodos.
h. **Árboles binarios de búsqueda** (ABB): son árboles binarios ordenados. Desde cada nodo todos los nodos de una rama serán mayores, según la norma que se haya seguido para ordenar el árbol, y los de la otra rama serán menores.
i. **Árboles AVL**: son también árboles de búsqueda, pero su estructura está más optimizada para reducir los tiempos de búsqueda.
j. **Árboles B**: son estructuras más complejas, aunque también se trata de árboles de búsqueda, están mucho más optimizados que los anteriores.
k. **Tablas HASH**: son estructuras auxiliares para ordenar listas.
l. **Grafos**: es el siguiente nivel de complejidad, podemos considerar estas estructuras como árboles no jerarquizados.
m. **Diccionarios**.

Criterios de selección

1. Priorizar de ser posible acceso directo y velocidad de procesamiento.
    a. Vector en primer lugar si se cumple tamaño fijo, razonable y conocido a priori, y sin necesidad de persistencia→ acceso directo, búsqueda binaria, búsqueda secuencial
2. Si no se conoce el tamaño y no se requiere persistencia
    a. Estructuras enlazadas
        i. Pila si se debe invertir el orden o si es irrelevante
        ii. Colas si se debe mantener
        iii. Listas si se debe generar
3. Archivo si se requiere persistencia: directamente o estructuras auxiliares después cargar al archivo

Toma de decisiones

1. Los datos se ingresan desde el teclado
    a. Se deben mostrar en la misma secuencia de entrada: No es necesario guardarlos en una estructura auxiliar. Así como los recibimos se deben mostrar, no se requiere conservarlos en memoria para ningún procesamiento posterior.
    b. Se requiere procesamiento posterior. Supongamos que debemos mostrarlos ordenados por un criterio diferente al ingreso: en este caso se debe generar una estructura auxiliar, esta puede ser un vector o lista si el ordenamiento es por un campo¸ matriz, vector de listas o lista de listas si el ordenamiento es por dos criterios
2. Los datos se ingresan desde un archivo físico
    a. Similar al ingreso por teclado → solo cambia el origen del dato
    b. Similar al ingreso por teclado → solo cambia el origen del dato
3. Secuencia de decisiones
    a. Origen del dato
        i. Teclado
        ii. Archivo
    b. Elección de estructura auxiliar
        i. Solo se muestra con el criterio de ingreso; no se requiere estructura auxiiiar
        ii. Se requiere reordenar o conservar para buscar: Requiere auxiliar

1. Orden por un campo el tamaño es conocido a priori
    a. Vector
        i. Carga directa
        ii. Carga secuencial
            1. Ordenar posteriormente
            2. Cargar ordenada
            3. Dejar sin orden, búsqueda secuencial
2. Orden por un campo tamaño no conocido
    a. Lista ordenada
        i. Insertar ordenado
        ii. CargarSinRepetir
        iii. InsertarOrdenado
3. Orden por dos campos
    a. Ambos conocidos y acotados
        i. Vector de vector → matriz
    b. Uno acotado y definido, el otro no
        i. Vector de punteros
    c. Ambos No acotados
        i. Lista de listas

c. Que guardar en la estructura auxiliar
    i. Todos los datos si se requieren
    ii. Selo los que se requieren como salida, no hacer substancia de lo superfluo
    iii. La clave de ordenamiento y la referencia al dato

Operadores de C

Símbolo [1]	Tipo de operación	asociatividad
`[ ] ( ) . ->` `++--` (postfijo)	Expresión	De izquierda a derecha
`sizeof & * + - ~ !` `++--` (prefijo)	Unario	De derecha a izquierda
typecasts	Unario	De derecha a izquierda
`* / %`	Multiplicativo	De izquierda a derecha
`+ -`	Aditivo	De izquierda a derecha
`<< >>`	Desplazamiento bit a bit	De izquierda a derecha
`< > <= >=`	Relacional	De izquierda a derecha
`== !=`	Igualdad	De izquierda a derecha
`&`	AND bit a bit	De izquierda a derecha
`^`	OR exclusivo bit a bit	De izquierda a derecha
`|`	OR inclusivo bit a bit	De izquierda a derecha
`&&`	AND lógico	De izquierda a derecha
`||`	OR lógico	De izquierda a derecha
`? :`	Expresión condicional	De derecha a izquierda
`= *= /= %=` `+= -= <<= >>= &=` `^= |=`	Asignación simple y compuesta [2]	De derecha a izquierda
`,`	Evaluación secuencial	De izquierda a derecha

Declaraciones de variables simples

```
int x, y;         /* Declara dos variables simples de tipo int */
int const z = 1; /* Declara una constante z de tipo entero y valor 1 */
```

Alcance y visibilidad de los identificadores

Esta propiedad refiere a su visibilidad o “reconocimiento” dentro de la aplicacion. Puede ser un alcance global, que es reconocida en toda la aplicación o local propia de cada modulo o, inclusive, bloque en el que esta declarado

Fundamentos C++

Sin declaración using	Con declaración using

//programa para imprimir texto
#include <iostream>
 	
int main(){
  std::cout << “Hola\n”;
  return 0;
}
	//programa para imprimir texto
	#include <iostream>
	using std::cout; //
	using std::cin;
	using std::endl;
	int main(){
	  cout << “Hola” << endl;
	  return 0;
	}

Instruccion	Descripcion
#include	Directiva del preprocesadpr
<iostream>	Componente de entrada/salida (objetos cin, cout,cerr)
using	Declaración que elimina necesidad de repetir el prefijo std.
int main()	Funcion principal que retorna un entero
{ }	Definición de un bloque de programa
std::cout	Uso del nombre cout del espacio de nombres std, dispositivo std de salida
::	Operador binario de resolución de alcance
<<	Operador de inserción en flujo
“Hola\n”	Literal Hola + salto de línea (también << std::endl;
;	Finalización de una sentencia
return 0	Punto de finalización correcta de la función

Instrucción	Descripcion
Cin	Dispositivo std de entrada
>>	Operador de extracción de flujo
+	Operador de suma
-	Operador de resta
*	Operador multiplicativo
/	Operador de división
%	Operador de modulo o resto
( )	Operador para agrupar expresiones ej: a * (b+c)
==	Operador de igualdad
>	Mayor
>=	Mayor igual
<	Menor
>=	Menor igual
!=	Operador de desigualdad
=	Operador de asignación
+=	Asignación y suma x+=3; equivale a x = x + 3;
-=	Resta y asignación
*=	Multiplicación y asignación
/=	División y asignación
++	Operador de incremento
--	Operador de decremento

Concepto de biblioteca
Al incluirlas se pueden utilizar todas las funciones que contienen sin necesidad de una nueva definición
#include <nombre de la biblioteca>
Iostream Es la biblioteca que debe estar presente siempre en todas las aplicaciones de la materia Está especializada en la lectura y escritura y es exclusiva de C++.

Espacios de nombre
Es una región declarativa que proporciona un ámbito a los identificadores (nombres de tipos, funciones, variables, etc.) de su interior.

Proposito de las funciones

1. Permite la descomposición como forma de alcanzar la solución
    a. Si L = L1 + L2 → Esf(L) > Esf(L1) + Esf(L2)
2. Promueve la modularidad
3. Favorece
    a. Comprension
    b. Trabajo en equipo
4. Facilita el codigo
    a. Evita repeticiones
5. Perimte
    a. Integridad y protección del dato
    b. Reusabilidad → uso de parametros
    c. Separar la lógica de la algoritmia → funciones de criterio
    d. Separar la lógica del tipo de dato → plantillas

Invocacion/declaración → distintos esquemas

Definicion anticipada	Declaracion definición	Bibliotecas propias

Definición de la función
int f(int a, int b){
return a + b;
};
int main()
{ ……..
Invocación
10 5
x = f(y,z);argumentos
f(x,y)
w = f (3*2,4);
….
}
	Declaración de la firma
	int f(int , int);
	
	int main()
	{ ……..
	invocación
	x = f(y,z);;
	…
	w = f (3,4);
	….
	}
	Definicion de la funcion
	int f(int a, int b){
	return a + b;
	};
		Inclusión de la biblioteca
		#include “miBiblio.h”
		
		int main()
		{ ……..
		
		invocación
		x = f(y,z);;
		…
		w = f (3,4);
		….
		}
		La biblioteca separa los prototipos en un archivo .h y el código en un archivo .c

Declaracion de una funcion

Firma o prototipo

Tipo de retorno nombre (lista de tipo de dato del parametro y tipo de parametro)
int suma(int, int);

1. Tipos de retorno
    a. Escalares
    b. Srtruct
    c. Punteros
    d. Valor ausente (void)
2. Nombre → regla identificadores Letra(Letra+Digito)*
3. Argumentos
    a. Por valor tipo
    b. Por referencia o dirección tipo&

Definicion de una función

Tipo de retorno nombre (lista parámetros su tipo y tipo de dato){
int suma(int a, int b){
Cuerpo de la función
}

1. Tipos de retorno
    a. Escalares
    b. Srtruct
    c. Punteros
    d. Valor ausente (void)
2. Nombre → regla identificadores L(L+D)*
3. Parametros
    e. Por valor tipo identificador
    f. Por referencia o dirección tipo& identificador
2. Cuerpo de la función
    {
    Declaraciones locales;
    Acciones;
    Retorno;
    }

Invocaciones

1. Con valor ausente una invocación a si misma
    a. Nombre(lista de argumentos)
2. Retornando escalar, puntero o struct en una expresión
    a. Identificador = Nombre(lista de argumentos)
3. Los argumentos vinculados con parámetros variables deben ser valorL
4. Los argumentos vinculados con parámetro valor son expresiones, completas o incompletas
5. C++ pasa parámetros por valor o referencia, eso se evidencia en la declaración y definición pero no en la invocacion

Ejemplos

Funcion	Detalle
Firmas o prototipos	
void F1(int)	Firma de función que no retorna valor y recibe un parámetro por valor
void F2(int&)	No retorna valor pero evalua y modifica el argumento vinculado al parámetro
int F3(int)	Similar a F1 retornando un escalar
int F4(int&)	Similar a F2 retornando un escalar
int F5(int, int)	Evaluando dos parametros valor
int F6(int, int&)	Evaluando un parámetro por valor y otro por referencia
int F7(void)	Declaración explicita sin parámetros
Invocacionei	
F1(3*6) F1(a) F1(a+b)	Invocado con una expresión completa vinculada al parámetro valor Imvocado con un identificador (expresión incompleta)
F2 (a)	Invocado con un valor por el ásaje por referencia
A = F3(2*4) A = F3(b)	Un parámetro valor
A = F4(X)	Un parámetro por referencia
A = F5(x, 1+2)	Dos parametros valor
A = F6(14, w)	Un parámetro valor y otro por referencia
A = F7()	Sin parámetros

Pasaje de cadenas como parametros

int F (char* s) o int F (char s[])
Pasaje de vectores y matrices
int F (int v[])
int F (int m[][columnas]

Reusabilidad ejemplos

A = suma1(B, C);
D = suma1(D, E);
F = suma1 (2*3, 4-2);

Generalidad

1. Según el tipo de dato → plantillas
2. Según el criterio de selección → punteros a funciones

Generalidad por tipo de dato	
Sin generalizar	**Generalidad con plantilla**

int suma1 (int a, int b){
int c = a+b;
return c;
}
 	
float suma1 (float a, float b){
float c = a+b;
return c;
}
 	
Invocación
X = suma1 (y,z)
	template < typename T>
	T suma1 (T a, T b){
	return a + b;
	}
	
	Invocación
	X = suma1<int> (y,z)
	X = suma1<float>(y,z)

Registros struct en C, C++

Registro: Es un conjunto de valores que tiene las siguientes características:

```
struct NombreTipo {
        Tipo Identificador;
        Tipo Identificador;
}
struct TipoRegistro {
        int N;
        float Y;
};                          //      declara un tipo
TipoRegistro Registro, r2; //       define una variable

struct TipoFecha {
        int     D;
        int     M;
        int     A;
};                          //      declara un tipo fecha

struct TipoAlumno {
        int             Legajo;
        char            Nombre[21];
        TipoFecha       Fecha
};                          //      declara un tipo Alumno con un campo de tipo Fecha

TipoAlumno Alumno;    //        define un identificador con la estructura declarada.
```

legajo	Nombre	Fecha		
int	Cadena	D int	M int	A int

En el caso de la definición precedente, Alumno es un registro (struct para C) con tres miembros (campos) uno de los cuales es un registro (struct) de TipoFecha. El acceso es:

Nombre	Tipo dato	
Alumno	Registro	Registro total del alumno
Alumno.Legajo	Entero	Campo legajo del registro alumno que es un entero
Alumno.Nombre	Cadena	Campo nombre del registro alumno que es una cadena
Alumno.Fecha	Registro	Campo fecha del registro alumno que es un registro
Alumno.Nombre[0]	char	Carácter ´pos 0 de la cadena Npmbre
Alumno.Fecha.D	Entero	Campo dia del registro fecha que es un entero
Alumno.Fecha.M	Entero	Campo mes del registro fecha que es un entero
Alumno.fecha.A	Entero	Campo anio del registro alumno que es un entero

Arreglo: Colección ordenada e indexada de elementos con las siguientes características:
1 indice (1 dim) → vector → int v[5]
2 indices int m[5][5]

Busqueda lineal en un vector

```
#define TOPE 8

int busquedaSecuencial(int  buscado, int v[], int N)
{
        int i = 0;
        while(i<N&&v[i]!=buscado)
                i++;
        if(i<N) return i;
        return -1;
};
```

```
void cargarAlFinal(int b, int v[].int &N)
{ //control de tope antes de invocar a la función
    v[N] = b;
    N++;
    return;
  }
```

Busqueda directa PUP, clave posicional
saber nombre del equipo Numero 101 → V[101-101].nombre →V[0].nombre
saber nombre del equipo Numero 104 → V[104-101].nombre →V[3].nombre

Carga

1. **Declaración**
    a. int v[8] = {1,4,2,6,8,9,2,0};→ inicializa todas las posiciones con la lista
    b. int v[8] = {2,6,8};→ inicializa todas las posiciones, tres con la lista el resto con cero.
    c. int v[8] = {0} → inicializa todas las posiciones con cero
    d. equipo equipos[8] = ???? {{101 , “huracán”} , { , } , { , } }

2. **Secuencial → elemento por elemento**
    a. **asignación interna** → for(i=0;i<8;i++) V[i] = valor;
    b. **asignación externa** → for(i=0;i<8;i++) cin>> V[i];
3. **Directa** → Clave numérica, PUP→ pos V[Clave-valor posición inicial]
4. **ordenada o no sin repetición de clave**

búsqueda

1. **Directa** Clave numérica, PUP→ Ef(1)→ V[Clave – valor 1ra posición]

2. **Secuencial** {} valores sin orden → recorrer desde la primera posición hasta que lo encuentre o hasta que se termine el vector→ Ef(N)

```
int búsquedasecuencial(int v[], int buscado, int n){
vector, valor buscado, tope del vector
i = 0; → índice para recorrer desde el principio
while(i<n&&buscado!= v[i])
        i++; → para acceder a la sgte posicion

if(i<n) return i;
else return -1;//arbitrario indica dato buscado no esta
retorna la posicion donde el buscado esta o -1 si el buscado no esta
}
```

3. **Binaria → deben estar ordenados**→ Ef(logarítmica)

```
int busquedaBinaria(int v[], int buscado, int N, int& primero){
//retorna la posición donde está el dato o menos uno si no lo encuentra
        int ultimo = N-1;
        int medio;
        primero = 0;
        while (primero <= ultimo){
                medio = (primero + ultimo)/2;
                if(v[medio] == buscado) return medio; //la posición donde lo encontró
                if(buscado > v[medio]) primero = medio + 1;
                else ultimo = medio – 1;
        };
        return -1;// si sale del while es que el primero es mayor que ultimo ese es el indicio que el dato
buscado no esta por eso retorna -1
}
```

```
void ordenarVector(int v[], int N) {// v el vector a ordenar N cantidad de componentes
        int i,j, aux;
        for(i = 1; i < N; i++){// pasos → 1..N-1
                for( j = 1; j <= N – i;j++){//comparaciones en cada paso → 1..N-i
                        if(v[j-1] > v[j]{
                                aux = v[j];
                                v[j] = v[j -1];
                                v[j – 1] = aux;
                        }
                } // fin ciclo interno
        }// fin ciclo externo
return;
} // fin de la funcion
```

Recorridos

1. Completo
   fo(i = 0; i<N, i++)
   cout << v[i];

2. Con corte de control
   Precondición → Al menos un campo que se repite, agrupado por ese campo, que conforman un subconjunto
   Pos condición → Información de cada subconjunto y/o del conjunto general
3. Apareo
   Precondición → Al menos dos estructuras con al menos un campo en común y ordenados.
   Pos condición → manejo conjunto intercalando y manteniendo el orden

Ordenar por más de un campo → solo cambia el criterio de selección

	c1	**c2**
0	20	11
1	8	2
2	15	18
3	20	6
4	3	11
5	3	3
6	20	4
7	15	9

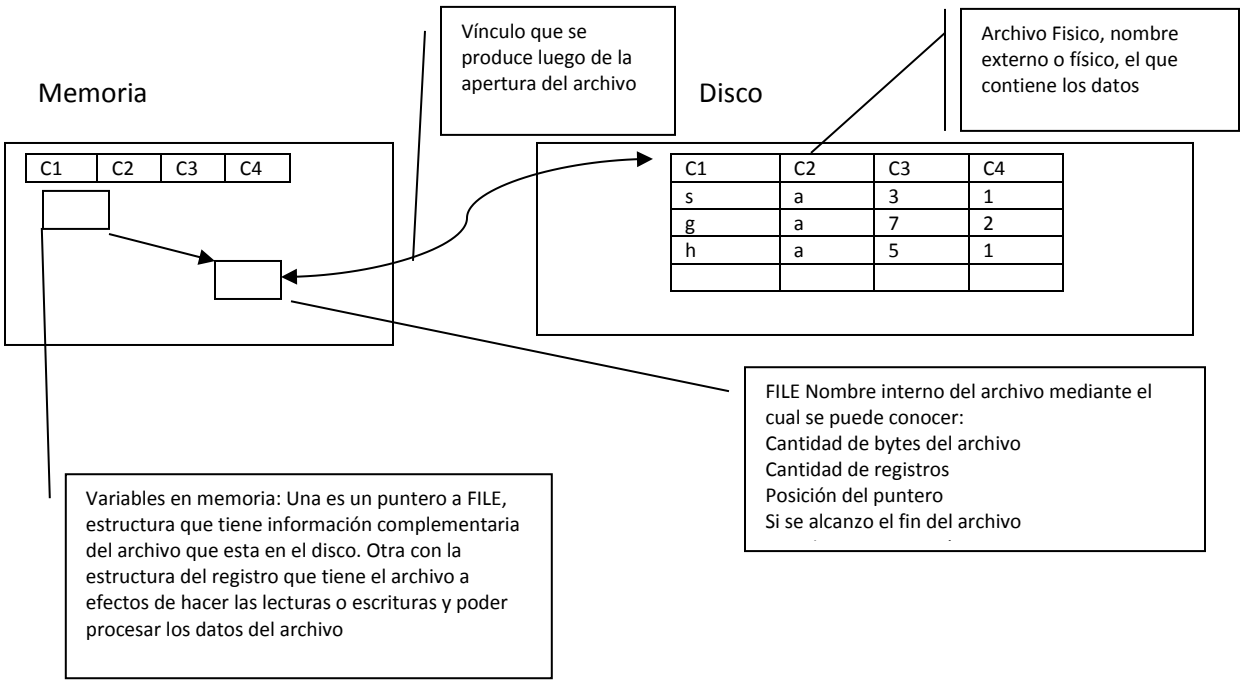
0	3	3
1	3	11
2	8	2
3	15	9
4	15	18
5	20	4
6	20	6
7	20	11

Cambiar la lógica Vs cambiar la algoritmia

```
void ordenarVector(int v[], int N) {// v el vector a ordenar N cantidad de componentes
        int i,j, aux;
        for(i = 1; i < N; i++){
                for( j = 1; j <= N – i;j++){
                        if(v[j-1].c1 > v[j].c1 || (v[j-1].c1== v[j].c1&& v[j-1].c2> v[j].c2)){
                                aux = v[j];
                                v[j] = v[j -1];
                                v[j – 1] = aux;
                        }
                } // fin ciclo interno
        }// fin ciclo externo
return;
} // fin de la función
```

Estructura tipo Archivo

- Archivos
  - De texto
  - Binarios
    - De tipo
      - De tipo registro con registros de tamaño fijo
    - Con acceso directo
    - En la implementación en C utilizamos
      - FILE *
      - fopen
      - fread
      - fwrite
      - feof
      - fseek
      - ftell
      - fclose



Archivos binarios: Analisis, síntesis y comparaciones

En la implementación en C utilizamos

FILE* se asocia a un flujo para controlarlo→ FILE* f;

fopen asocia el nombre lógico al físico→ f=fopen("NombreFisico","rb+");

fread lectura por bloques→fread(&r, sizeof(r), 1, f);

fwrite escritura por bloques→fwite(&r, sizeof(r), 1, f);

feof indicador de final del archivo→feof(f)

fseek permite acceso directo→ fseek(f,2*sizeof(r),SEEK_SET)

ftell retorna de bytes de desplazamiento desde el inicio del flujo→ftell(f);

fclose cierra el archivo, vacía el buffer y coloca marca de EOF→flose(f)

struct tr{ int c1; int c2;} → supongo entero de 8 bytes

8		16		24		32		40	
3	5	4	1	2	8	14	5	5	11

ftell(f) →8 24 36

fseek(f, 16, SEEK_SET) →

fseek(f, 2*8, SEEK_CUR) →

fseek(f, -sizeof(r), SEEK_END) ←

ftell(f)/sizeof(r) → 2 → 4

fseek(f,0,SEEK_END); p = ftell(f)/sizeof(r);→calcula cantidad de registros

tr v[10]

FILE* f = fopen("ma","rb+")

a=fread(&reg, sizeof(reg),1,f)

3	5

fread(&a, sizeof(int),1,f)

3

fread(v, sizeof(reg),4,f) v → vector de struct; v[0] → struct pos 0

read(v, sizeof(reg),4,f)

fread(&v[1], sizeof(reg),4,f)

Análisis comparativo Archivos Vectores→Declaración, Acceso, busqueda

Vector (de registros)	Archivo (de registros)
Consideraciones generales	
Almacenamiento lógico (memoria) Tamaño Fijo (T. E.). Procesamiento rápido Sin persistencia después aplicación Prioriza velocidad procesamiento	Almacenamiento físico (disco) Tamaño variable (T.E.). Procesamiento lento Con persistencia después aplicación Garantiza persistencia
Definiciones y declaraciones	
Tr V[X]; Note que determina a priori el tamaño (X) y especifica el tipo de dato de cada posición(Tr).	FILE* f = fopen (“XXXX”, “rb+”); Solo indica como lo abre sin especificar particularidad del dato, solo “b o t” y no el tamaño
Acceder al registro de posición N	

V[N];
Accede al registro en memoria con esa posición.
	fseek(f, N*sizeof(r), SEEK_SET);
	posiciona el puntero en el registro
	fread(&r, sizeof(r), 1, f);
	Lleva a memoria el registro N.

Modificar el registro de la posición N	

V[N].campo = valor
Modifica el campo especifico del registro N que está en memoria
	fseek(f, N*sizeof(r), SEEK_SET);
	APUNTAR al registro a modificar
	fread(&r, sizeof(r), 1, f);
	LEER Lo lleva a memoria.
	r.campo = valor
	MODIFICAR el dato en memoria
	fseek(f, N*sizeof(r), SEEK_SET);
	Vuelve APUNTAR al registro.
	fwrite(&r, sizeof(r), 1, f);
	GRABAR en el disco.

Acceder al registro siguiente al de la posición N y tenerlo a disposición	

V[N+1]	fseek(f, (N+1)*sizeof(r), SEEK_SET);
	APUNTAR al registro a modificar
	fread(&r, sizeof(r), 1, f);
	LEER Lo lleva a memoria.

Acceder al primer registro y tenerlo a disposición	
V[0]	fseek(f, 0, SEEK_SET); fread(&r, sizeof(r), 1, f);

Acceder al último registro y tenerlo a disposición	
V[X-1]	fseek(f, -sizeof(r), SEEK_END); fread(&r, sizeof(r), 1, f);
Búsqueda binaria	

int bb(Tr V[], int X, int N){
int p = 0;
int u = N-1;
int m;
 	
while(p<=u){
m = (p + u)/2;
 	
if(V[m]. campo == X) return m;
else if(X>V[m].campo p= m++;
else u = m--;
}
return -1;
}
	int bb(FILE* f,int X){
	int p = 0;
	int u = cantRegistros(f)-1;
	int m;
	Tr r;
	while(p<=u){
	m = (p + u)/2;
	fseek(f,m*sizeof(r), SEEK_SET);
	fread(&r, sizeof(r), 1, f);
	if(r. campo == X) return m;
	else if(X>r.campo p= m++;
	else u = m--;
	}
	return -1;
	}

Búsqueda directa (PUP)	
V[Clave-vInicial]	fseek(f,sizeof(r)*(Clave-vInicial),SEEK_SET) APUNTAR al registro buscado fread(&r, sizeof(r), 1, f); LEER el registro apuntado
Recorrido	

i= 0;// ir al inicio
 	
while(i<N) { //control de fin.
procesar v[i];
i++; //avanzar
}// fin del ciclo
	fopen pone epuntero al inicio
	Opcion 1 lectura anticipada
	fread(&r, sizeof(r), 1,f); // leer
	while(!feof(f)){
	procesar r;
	fread(&r, sizeof(r), 1,f);
	}
	Opcion 2 lectura y verificacion simultaneal
	while (fread(&r, sizeof(r), 1, f)){
	procesar r;
	}

<table>
<tr><th colspan="2">Apareo (conceptual)</th></tr>
<tr><td>Concepto: Aplicable a dos o más estructuras son al menos un campo en común ordenadas por ese campo que se procesan paralelamente, intercalando los valores para conservar el orden</td><td>Seudocódigo:
Situarse al principio de ambas estructuras
Hacer mientras (haya datos en ambas)
Si la primera cumple criterio de ordenamiento
Procesarla;
Avanzar con ella
Sino
Procesar la otra;
Avanzar con ella
Fin del mientras
Agotar la estructura que no se termino
Procesarla;
Avanzar
Fin del proceso</td></tr>
<tr><th>Vectores</th><th>Archivos</th></tr>
<tr><td>int i = 0;
int j = 0;

while((i<N) && (j<M)){
if(v1[i].c1<v2[j].c1){
//procesar v1[i];
i++;}
else{
//procesar v2[j];
j++;}
}
while(i<N){
//procesar v1[i];
i++; }
while(j<M){
//procesar v2[j];
j++;}</td><td>fread(&r1,sizeof(r1), 1, f1);
fread(&r2,sizeof(r2), 1, f2);

while(!feof(f1)&& !feof(f2)){
if(r1.c<r2.c){
// procesar r1;
fread(&r1,sizeof(r1), 1, f1)};
else{
// procesar r2;
fread(&r2,sizeof(r2), 1, f2) ;}
};
while(!feof(f1)){
procesar r1;
fread(&r1,sizeof(r1), 1, f1)};
while(!feof(f2)){
procesar r2;
fread(&r2,sizeof(r2), 1, f2)};</td></tr>
</table>

1
8
14
22
125

2
3
11

```
int j = 0; int i = 0;
// j==M || i<N && v1[i].c < v2[j].c
while((i<N) || (j<M)){
                if(j==M || i<N && v1[i].c < v2[j].c){
                   //procesar v1[i];
                   i++;}
                else{
                   //procesar v2[j];
                   j++;}
```

Punteros y estructuras enlazadas lineales

Puntero

Valores → Direcciones de memoria donde un dato o código

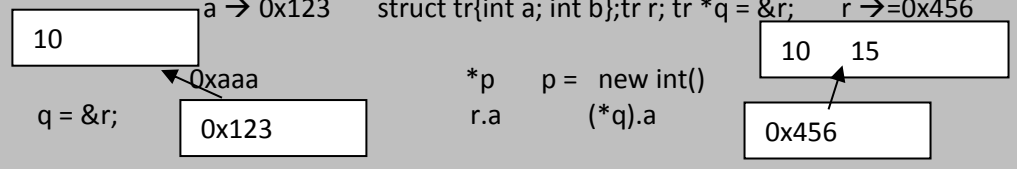
Operaciones → asignación →&, new(tipo) NULL → liberar memoria delete puntero;

Ejemplo int* p;

int a = 10; // declaración de una variable de tipo entero inicializada en 10

int *p=NULL,*p2;// declaración de puntero a

a → 0x123 struct tr{int a; int b};tr r; tr *q = &r; r →=0x456



p = &a; *p

Nombre	tipo de dato	valor	
a	int	10	q=new tr();
p	puntero a int	0x123	delete (p)
*p	int	10 → valor contenido en el lugar de memoria donde apunta p	
r	tr	10, 15	r.a r.b
q	puntero a tr	0x456	
*q	tr	10, 15	(*q).a q->a (*q).b q->b
(*q).b	int	15 (*q).a	
q->b	int ->	15 →operador de acceso	

Estructuras enlazadas

Nombre	Cargan	sacan
Pilas	Delante del primero	De la primera posición
Colas	Después del ultimo Prioridad	Primera posición Según la prioridad
Listas simplemente enlazadas Listas circulares Listas doblemente enlazadas Listas doblemente enlazadas	Según criterio	Según ese criterio

```
void push(Nodo* & Pila, int x){        se invoca con la pila, y el valor a guardar x
 Nodo*p = new Nodo();                  pedir memoria
  p→info = x;                          guardar la informacion
  p→sgte =Pila;                        enlazar el nuevo nodo con la est. existente delante de Pila
  Pila = p;                            el nodo creado pasa a ser el primero
 return;
}
```

En el caso de necesitar eliminar un nodo de la pila y retorna el valor allí guardado la secuencia será:

```
int pop(Nodo* &pila){   recibe la pila como parámetro y retorna el valor contenido en el primer
nodo
  int x;                el tipo de dato de la información para retornarlo
  Nodo* p = Pila;       un puntero al comienzo de la estructura para luego eliminar ese nodo
  x = pila→info;        conservar el valor del primer nodo para retornarlo
  pila = p→sgte;        Avanzar con la pila un nodo para eliminar el nodo que estaba en el tope
  delete p;             eliminar el que era primero, la pila esta apuntando al que era segundo o a
NULL
  return x;             retornar la información que estaba en el primer nodo
}
```

Estructura tipo cola

con dato de tipo simple

```
void queue(Nodo*&frente, Nodo* &fin, int X){
   Nodo * p = new Nodo();
   p→info = x;
   p→sgte = NULL;
   if (frente==NULL) frente = p; else fin→sgte = p;
   fin = p; se redefine el nuevo fin apuntando a este nuevo nodo
   return;
}
```

con dato de tipo simple

```
int unqueue(Nodo*&frente, Nodo* &fin){
   int x;
   Nodo * p = frente;
   x = p→info;
   frente = p→sgte;
   if (frente==NULL) fin = NULL;
   delete p;
   return x;
}
```

Procedimientos de listas

```
Nodo* buscar(Nodo* l, int v){
```

C1	C2	C3
0 1		
1 4		
2 3		
3 7		
4 5		

```
  Nodo* aux = l;
while( aux!=NULL && aux->info.c <v.c )   {
    aux=aux->sig;
  }
  return (aux!=NULL&&aux->info.c==v.c)?aux:NULL);
}
//Exp?a1:a2
Nodo* InsertarOrdenado(Nodo* & Lista, int/TipoInfo x){
        Nodo* Nuevo = new Nodo();
        Nuevo ->info = x;
        if(Lista == NULL || x.c1<Lista->info.c1)
                {
                        Nuevo->sgte = Lista;
                        Lista = Nuevo;
                }
        else    {
                        Nodo * p = Lista;
                        while(p->sgte!=NULL &&  x.c > p->sgte->info.c)
                                p = p->sgte;
                        Nuevo->sgte = p->sgte;
                        p->sgte = Nuevo;
                }
        return Nuevo;
}
```

```
Nodo* InsertarSinRepetir(Nodo* &lista,int x){
   Nodo*p = buscar(Lista,x);
   if(p==NULL) p=InsertarOrdenado(lista, x);
   return p;
}
```

Ejemplos para reordenar un archiovo

1) Suponiendo los datos en archivo sin orden y se debe ordenar

guardar los datos en un vector→una clave conocida y posicional, la otra no

todos los datos

posición del archivo clave de busqueda

ordenar vector

mostrar los datos del vector

todos los datos → mostrarlos

busco en el vector → acceso directo al archivo a través de la posición

guardar los datos
i = 0;
while(fread(&r, sizeof(r), 1,f){
v[i] = r si guardo todos los datos
si solo quiero guardar la clave y el numero de registro
v[i].clave = r.c1;
v[i].pos = i;
i++;
}
ordenar vector
mostrar los datos según lo que se guarde
for (i=0, i<5;i++){
// si se guardaron los dtos
cout<<v[i].campo….
// si se guardo la clave y la referencia
fseek(f,v[i].pos*sizeof(r), SEEK_SET);→ acceso directo pos
fread(&r, sizeof(r), 1, f); → leer registro
cout<<r.c1……
}

→ si el tamaño se desconoce se guarda en una lista

i=0;
while(fread(&r, ……..)){
si son todos los datos x = r
si es parte de los datos x.c1=r.c1……. y lo que se requiera
si es la clave y la referencia x.c1=r.c1; x.pos = i
i++;
insertarOrdenado(lista, x);
}

//
Mostrar los datos
while(lista != NULL){
x=pop(lista);
// si el nodo tiene todos los datos
cout<<x.c1<<……
//si en el nodo esta la clave y la referencia
fseek(f,sizeof(r)*x.pos,SEEK_SET)
leer el registro
mostrar los datos
}

Ordenar por dos campos en una matriz

C1	C2	Nombre
1	5	Hola
4	3	Chau
1	8	Huracan
7	6	Utn
4	1	Ceit

c1 1..10
c2 1..30
matriz
string matriz[10][30]
carga matriz
while(fread(&r, sizeof(r), 1,f){
matriz[r.c1-1][r.c2-1]=r.nombre;
}
mostrar
for(fila=0, fila < 10, fila++)
for(col=0, col< 30, col++)
cout<<m[i][j]

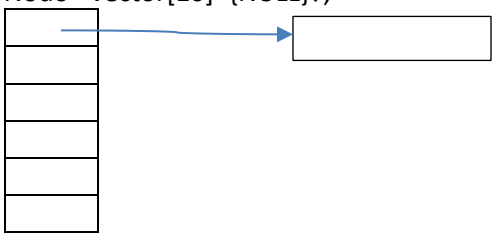
en un vector de punteros
c1 1..10 clave acotada y consecutiva
c2 entero
vector de punteros
struct tipoinfo{
int c2;
string nombre;
}

C2 int	Nombre string

struct Nodo{
tipoinfo info;
Nodo* sgte;
}

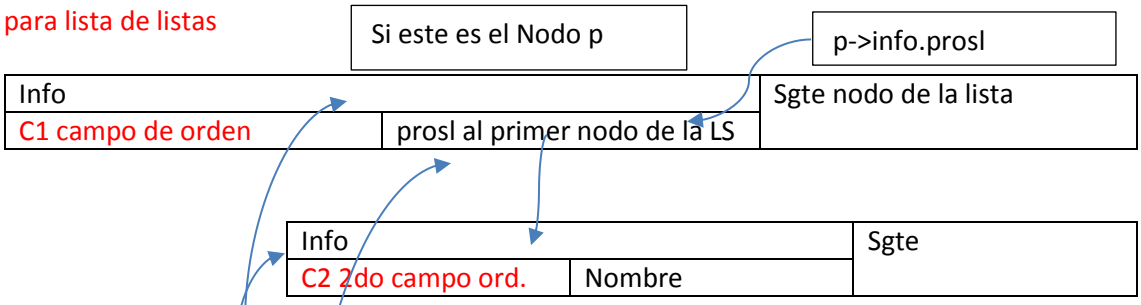
Info		Sgte
C2	Nombre	

tipoinfo valor;
Nodo* vector[10]={NULL}?;



```
//inicializar los punteros a las 10 listas
for(i=0, i < 10, i++)
vector[i] = NULL;

cargar  vector de listas
while(fread(&r, sizeof(r), 1,f){
 valor.c2 = r.c2;
 valor.nombre = r.nombre;
 insertarordenado(vector[r.c1-1], valor);
}
for(i=0, i < 10, i++)[{
 while(vector[i]!=NULL){
      valor = pop(vector[i]); cout<<valor.c2……;}
```

c1 int, c2 int no acotados → lista de lista si el propósito es ordenar por dos campos
en este caso se utiliza cargar sin repetir en combinacion con otra forma de insercion según el contexto
cargar sir repetir tiene dos aplicaciones fundamentales, una para acumular según una clave y la otra para lista de listas

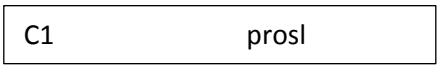



```
struct NodoLS{
tipoinfo info;
NodoLS* sgte;
};
tipoinfo valorLS;
struct TipoinfoLP{
int c1; TipoinfoLP info;

 NodoLS* prosl;
};

struct NodoLP{
NodoLP* sgte;
};
TipoinfoLP valorLP;
```

```
struct TipoInfoLP{
        int c1;
        NodoLS* prosl;
};
```

cargar los datos en una lista con sublista

```
while(fread(&r, sizeof(r), 1,f){
// cargar los registros de las listas
valorLS.c2 = r.c2; valorLS.nombre = r.nombre;
valorLP.c1 = r.c1; valorLP.prosl = NULL;
NodoLP * p=cargarsinrepetir(lista, valorLP);
insertarordenado(p->info.prosl  , valorLS);
};

while(lista!=NUll){
 valorLP = pop(lista);
        while(valorLP.prosl!=NULL){
        valorLS =  pop(valorLP.prosl):
        cout<< valorLS.c2.....
}
```



Como pensar los problemas de Algoritmos

Estructuras de datos: Características

	Vector	Archivo	Listas ordenadas
Almacenamiento	logico	fisico	Logico
Procesamiento	rapido	Lento	Rapido
Tamaño en T.E.	Fijo	Variable	Variable
Tamaño cada posicion	Informacion	Informacion	Info + sgte
Busqueda directa	SI→V[N]	fseek	Buscar secuencialmente
Persistencia	NO	SI	NO
Busqueda binaria	SI	SI	NO→arbol
Busqueda secuencial	SI	NO se recomienda	SI→unica posible
Carga directa	SI →V[N] = valor	Acceder → fseek Grabar → frwrte	Utilizando diferentes funciones
Carga secuencial	SI→v[ultimaPos] = valor	Acces. a ultima pos grabar	SI Insertar ordenado
Carga sin repetir	Buscar No esta agregar	NO se recomienda Est. Auxiliares	Buscar No esta agregar
Ordenamiento	PUP Metodos de ordenam.	PUP Solo usando est. aux	Genera insertarOrdenado
Definicion	Td Nombre[tamaño]	FILE*f; f= fopen(.....)	Definir Nodo → struct Punt Nodo* X = NULL
Acceso	V[N]	Acceder → fseek leerr → fread	Buscar secuencialmente
Corte de control	SI	SI	SI
Apareo	SI	SI	SI
Modificar la pos N	V[N] = valor	Apuntar → fseek Leer→fread Modificar en memoria Apuntar → fseek Grabar → fwrite	Buscar secuencialmente modificar

Estructuras de datos: Criterios de selección

4. Priorizar de ser posible acceso directo y velocidad de procesamiento.
    a. Vector en primer lugar si se cumple tamaño fijo, razonable y conocido a priori, y sin necesidad de persistencia→ acceso directo, búsqueda binaria, búsqueda secuencial
5. Si no se conoce el tamaño y no se requiere persistencia
    a. Estructuras enlazadas
        i. Pila si se debe invertir el orden o si es irrelevante
        ii. Colas si se debe mantener
        iii. Listas si se debe generar

6. Archivo si se requiere persistencia: directamente o estructuras auxiliares después cargar al archivo

Estructuras → muchos datos del mismo tipo

Archivos
Vectores → memoria → mejorar tiempos de procesamiento → tamaño fijo
Listas ordenadas → memoria →mejorar tiempos de procesamiento → tamaño variable

Observen como vienen los datos:

1. Es posible que los datos de partida estén en la forma en que nos piden los resultados→ procesamos directamente el archivo
2. Es posible que los datos de partida NO estén en la forma en que nos piden los resultados→ Debemos cargar en estructuras auxiliares y procesar esta estructura, reordenar los datos del archivo y procesar el archivo

Datos provienen de archivos

1. Tal como están se muestran

Consigna →Mostrar todos los datos del archivo
No requiere guardarlo en otra estructura, leer cada registro y mostrarlo

Dato Archivo ordenado

Consigna →Mostrar todos los datos del archivo ordenados por el mismo campo
No requiere guardarlo en otra estructura, leer cada registro y mostrarlo
Dado un archivo de registros mostrar su contenido por pantalla
Los resultados están en el orden que están en el archivo **No Necesitan estructura auxiliar**

2. Cambiar el orden, u ordenarlo

Consigna es mostrarlo ordenado, si esta desordenado, o el el campo no se corresponde con el resultado
Se requiere una estructura auxiliar memoria para facilitar el procesamiento
Vector → conozca tamaño a priori
Cargo el archivo al vector
Ordeno el vector
Muestro los datos del vector
Lista ordenada → No conozca tamaño a priori
Cargo el archivo a la lista ordenada
Muestro los datos de la lista

3. Ordenar un archivo desordenado

Dado un archivo de registros desordenado mostrar su contenido por pantalla ordenado por el campo1
Dado un archivo de registros desordenado ordenarlo por el campo1
Dado un archivo de registros desordenado mostrar su contenido por pantalla ordenado por el campo1 y a igualdad el campo 1 por el campo 2

Se requiere una estructura auxiliar memoria para facilitar el procesamiento
Vector → conozca tamaño a priori
Cargo el archivo al vector
Ordeno el vector
Me posiciono en la primera posición del archivo
Grabo los datos del vector → archivo se reemplaza y queda ordenado
Lista ordenada → No conozca tamaño a priori

Cargo el archivo a la lista ordenada
Me posiciono en la primera posición del archivo
Grabo los datos del vector → archivo se reemplaza y queda ordenado

---

Buscar en un archivo

1. Archivo ordenada tiene en la clave de búsqueda una PUP

No requiere estructura auxiliar → **búsqueda directa** Pos = Clave-valor 1ra pos
Si decido usarla puedo → vector No Lista→ solo permite secuencial
Clave 101 … 150 → buscar 135 → fseek(f, (135-101)*sizeof(r), SEEK_SET)

2. Archivo ordenado → la posicion no es predecible a priori

Una solución posible → **Busqueda binaria** en el archivo
Otra solución → cargarlo en estructura auxiliar y buscar en ella

3. Archivo desordenado

Necesariamente debemos cargarlo estructura auxiliar
Vector → si se da la condición del tamaño
Búsqueda secuencial
Ordenar el vector → búsqueda binaria → más eficiente que la secuencial
Cada posición solo necesitamos la información
Lista → si no se conoce el tamaño a priori
Búsqueda secuencial
Cada posición necesitamos la información + referencia al siguiente

Dados dos archivos uno con los datos personales de los alumnos, sin orden, el otro ordenado por legajo y materia con las materias cursadas Desarrollar un programa que muestre por pantalla los resultados de las materias y el nombre del estudiante. El campo que vincula ambos archivos es el numero de legajo

---

Datos están en dos estructuras y las quiero agrupar según un determinado orden

1. Dos archivos ordenados → intercalar conservando el orden

Apareo de los archivos sin necesidad de cargarlos en una estructura auxiliar

2. 1 archivo ordenado y el otro no

El ordenado lo dejo como esta
El desordenado lo cargo en una estructura auxiliar → vector ordenado o una lista ordenada
Una solución →apareo archivo ordenado con la estructura auxiliar
Otra solución → ordenar el archivo desordenado→ aparear los archivos

3. Ambos desordenados

Cargar ambos en una estructura auxiliar
Una solución → aparear las estructura auxiliares
Otra solución → ordenar los archivos → aparearlos
Otra solución → si alcanza → cargar ambos archivos misma estructura auxiliar y ordeno

Dados dos archivos ambos ordenados por numero de legajo, uno con los datos personales de los alumnos, el otro ordenado por legajo y materia con las materias cursadas Desarrollar un programa

que muestre por pantalla los resultados de las materias y el nombre del estudiante ordenado por numero de legajo. El campo que vincula ambos archivos es el numero de legajo

---

Porque y cuando corte de control

1. Datos en un archivo ordenado por dos campos (legajo y materia) con repetición del legajo

Mostrar los datos agrupados por el primer campo → no necesito estructura auxiliar

nl	Mat	nota
4	Algo	1
4	Ssl	2
7	Algo	2
7	Md	10
7	Syo	9

Mostrar los datos agrupados por nl → corte de control

NL 4
Mat nota
Algo 1
Ssl 2

Nl 7
Mat nota
Algo 2
Md 10
Syo 9

2. Archivo esta sin orden→ idéntica consigna

Cargarlo estructura auxiliar, ordenada por dos campos
Una solución→ hacer el corte de control en la estructura auxiliar
Otra solución → Ordenar el archivo, y corte de control en el archivo

Dni	Nl	cm	Nota

archivo

Dni	Nl	mat	nota
6	14	Alg	2
9	25	Sint	4
3	18	Syo	3
2	30	md	1
5	35	alg	3
14	24	md	1

Son 6 registros

```
struct tr{int dni; int nl; char cm[20]; int nota};
tr vector[6];
tr r;
```

vector

Dni	nl	mat	nota

```
FILE* f = fopen(“nombre”,”rb+”);
I = 0;
1 fread(&r,sizeof(tr),1,f);
while(!feof(f)){
 vector[i]= r;
i++;
 fread(&r,sizeof(r),1,f);
}

2 while(fread(&r,sizeof(r),1,f)){
 vector[i]= r;
i++;
}

3 for(i=0;i<6; i++){
fread(&r,sizeof(r),1,f);
vector[i]= r;
}

4 fread(&r,sizeof(r),1,f);
for(i=0;!feof(f); i++){
vector[i]= r;
fread(&r,sizeof(r),1,f);
```

}

5 fread(vector,sizeof(r),6,f);→ 5.1 fread(&vector[0],sizeof(r),6,f);

vector

Dni	nl	mat	nota
6	14	Alg	2
9	25	Sint	4
3	18	Syo	3
2	30	md	1
5	35	Alg	3
14	24	md	1

EN ESTE PUNTO VECTOR Y ARCHIVO DESORDENADO
ordenarVector(vector,6);// archivo sin orden, vector ordenado
Mostrar por pantalla sin modificar el archivo

```
for(i=0;i<6; i++)
cout <<vector[i].dni<< vector[i].nl……;
```

Reordenar el archivo

```
 fseek(f,0,SEEK_SET);
1 for(i=0;i<6; i++){
r = vector[i];
fwrite(&r,sizeof(r),1,f);
}

2 for(i=0;i<6; i++){
fwrite(&vector[i],sizeof(r),1,f);
}

3 fwrite(vector,sizeof(r),6,f);
```

Archivo ordenado

Volvemos al principio archivo sin orden y no se conoce el tamaño
Estructura adecuada lista

info				Sgte NODO*
dni	nl	mat	nota	

```
 struct Nodo{
          tr info;
          Nodo* sgte;
};
Nodo* Lista = NULL;

while(fread(&r,sizeof(r),1,f))
 insertarOrdenado(lista,r);
```

archivo sin orden. Pero la lista ordenada

mostrar fseek(f,o,SEEK_SET);

```
while(lista!=NULL){      ==
```

r = pop(lista); ==
cout<<r.dni……. fwrite(&r,sizeof(r),1,f)
}

A1

2
8
15

A2

3
4
5

R1

8

R2

5

A3

2
3
4
5
8
15

```
fread(&r1;sizeof(r1),1,A1);
fread(&r2;sizeof(r2),1,A2);
while(!feof(A1)&&!feof(A2)){
if(r1.nl<r2.nl){ fwrite(&r1;sizeof(r2),1,A3); fread(&r1;sizeof(r1),1,A1);}
 else fwrite(&r2;sizeof(r2),1,A3); fread(&r2;sizeof(r2),1,A2);}
}
// terminar el que no se agoto
while(!feof(A1)){
fwrite(&r1;sizeof(r2),1,A3);
fread(&r1;sizeof(r1),1,A1);
};

// terminar el que no se agoto
while(!feof(A2)){
fwrite(&r2;sizeof(r2),1,A3);
fread(&r2;sizeof(r1),1,A2);
}
```