UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE INGENIERIA INDUSTRIAL Y DE SISTEMAS

Curso: Lenguaje de Programación – SI401V

Escuela Profesional de Ingeniería Industrial

Ciclo: 2024-I

Material didáctico digital en software de apoyo académico para la enseñanza universitaria

Profesor Dr. Luis Lujan

2015, Doctor en medio ambiente y desarrollo sostenible, post doctoral.
2014, Doctor en ingeniería de sistemas, post grado.
2013, Master en ingeniería de sistemas, post grado.
1998, Ingeniero industrial.
Docente FIIS-UNI: post grado desde 2006 y pre grado desde 2003-II.
✉: llujanc@uni.edu.pe - luislujan@neosistemas.org
CV: http://www.neosistemas.org/l1.pdf

Inicio: 18 de Marzo del 2024 - Lima Perú

Índice

PRIMERA SEMANA:

Introducción.
Presentación de software en computadora. Importantes características del lenguaje:

- Compilador, permite generar archivo ejecutable.
- Óptimo para desarrollar: lenguajes, sistemas operativos, sistemas de gestión de bases de datos (de las siglas en ingles DBMS), utilitarios, entre otros, singularmente para diversas aplicaciones como sistemas de software como por ejemplo de contabilidad.
- Portable con pocas o ninguna modificación. Multiplataforma en software y hardware.
- De propósito general. Permite crear librerías para ser utilizadas por otros lenguajes.
- Es estructurado y orientado a objetos. Existen versiones de diversas empresas.
- Se usa como interface para lenguaje assembler y otros lenguajes.

Componentes de la estructura
Se compone de uno o más bloques de instrucciones.
La función main () es la función principal de un programa, es la que gobierna la ejecución, ejecuta funciones ó instrucciones según el orden en que aparecen en su bloque, algunas pueden ejecutar otras funciones.
Los comentarios no se compilan y se escriben anteponiendo //

Directivas de preprocesador
Cabeceras
También conocidas como 'cabeceras', son librerías de biblioteca que contienen funciones del sistema del lenguaje. Y para cada una se escriben con #include.

#include
Se utiliza para incluir el contenido de un archivo en otro archivo fuente antes de que el código se compile. Esto es útil para dividir un programa en varios archivos separados y reutilizar código.

Entre las más utilizadas tenemos:

```
#include <stdio.h>      //funciones de entrada y salida
#include "stdlib.h"     //funciones de bibliotecas más comunes
#include <math.h>       //funciones matemáticas
```

Otras: #include <conio.h> //usado para getch() en algunos codigos, similar a system("pause")
<time.h> //funciones de hora y fecha #include <string.h> //funciones relacionadas a cadenas

#define
Se utiliza para crear macros de reemplazo de texto. Una macro es un nombre que se asocia con un valor o con un fragmento de código. Cuando se utiliza #define, estás creando una macro que luego puede ser usada para reemplazar cualquier instancia de ese nombre en el código fuente por el valor o fragmento de código asociado.

```
#define pi 3.14159265
#define 0 'verdadero'
```

Estructuras
Hola mundo

```
#include <stdio.h>
main(void)
{
  printf("Hola mundo");
}
```

Estructura Básica

```
#include <stdio.h>      //"stdio.h" Cabecera definiciones estándares de entrada salida
main(void)              //Función principal - obligatorio
{                               //Inicio de bloque principal
  float x,y,z;          //Declaración de variables
  x = 1000.0;           //Asignación de variable
  y = 1003.0; z= x+y;
  printf("Suma : %d",z);        //Imprime resultado con función printf
}                               //Termina bloque principal
```

Estructura con funciones de usuario

```
#include <stdio.h>      //Cabecera definiciones estándares entrada salida
#include "stdio.h"      //Otra forma de escribir la anterior
```

```
                        //Las 2 barras indican una línea como comentario
int suma(int a,int b);  //Definición de una función suma
void main(void)         //Función principal - obligatorio
{                               //Inicio de bloque del programa
 int x,y,z;             //Declaración de variables
 x = 1000;              //Asignación de variable
 y = 1003;

 z=suma(x,y);           //Llamada a función suma
 printf("Suma : %d",z);         //Imprime resultado con función de biblioteca printf
}                               //Termina bloque del programa

                        //Las líneas en blanco se ignoran
int suma(int x,int y)   //Código de la función suma
{
 int z;                 /* ; después de cada “orden” */
 z =x+y;                        //  /* y */   otra forma de comentar, la anterior línea, se utiliza para varias líneas
 return z;
}
```

Otra estructura: ver clases de objetos.

Otras indicaciones:
En cualquier lugar se pueden incluir comentarios empleando los símbolos /* */ o //
El símbolo de punto y coma (;) se utiliza al finalizar una “orden/instrucción” (exceptuando for e if)
Un bloque encerrado por los símbolos { }, es tratado como una instrucción simple
Toda función forma un bloque de instrucciones.

Ejemplo:
int x = 2; /* esto es un comentario que será eliminado o sustituido por un simple espacio
Como puede verse, el comentario puede ocupar varias líneas de texto en el código fuente. */

Ejemplo, la expresión:
int /* declaración */ i /* contador */;

Se puede anidar bloques:

```
{ /*comienzo del bloque */
  ....
        { /*Comienzo del sub - bloque */
                ...
                ...

        } /*fin del sub - bloque */
  ....

} /*fin del bloque */
```

La ejecución del código es secuencial, salvo que alguna instrucción de estructura de control y otras.

Elementos del Lenguaje

Identificador

Se compone de uno o varios caracteres, es sensible a minúsculas/mayúsculas y se forma según:
1.El primer carácter debe ser una letra o el símbolo “_”
2.Los caracteres siguientes pueden ser letras, el símbolo “_” ó dígitos
3.Diferentes a las palabras reservadas a int, if, etc.
Se utilizan para definir Funciones de usuario, variables, constantes, etc.
Ejemplos: límite, valor1, suma_total, _mensaje, A, x3,
#define máximo 100 (directiva que define al identificador máximo como una constante 100), etc.

Constantes

Ejemplos:

```
const int X = 10;       // X es un tipo int constante
#define pi  3.1415
#define m “Mensaje: “
```

Errores por :

Asignación	int const k = 2+(k = 4);	// Error
Coma	int const k = (i=1, 3);	// Error
Decremento	int const k = k--;	// Error
Llamada a función	int const k = func(4);	// Error
Incremento	int const k = k++;	// Error

Tipos de datos

Float (**float**) Números reales, de punto flotante positivo, negativo o cero, con decimales.
Entero (**int**) Números enteros.

Carácter (**char**) Un carácter: un número, una letra o un símbolo
Precisión doble (**double**) Números reales, de punto flotante positivos, negativos o cero, con decimales que son representados con un número mayor de dígitos que los float, teniendo un mayor grado de precisión.
Ausencia de tipo dato (**void**) //En varios casos depende de la configuración del software del lenguaje

Tipos extendidos:
Son "adaptaciones" de detalle sobre los tipos básicos para mejor adaptarse a necesidades específicas.
largo (**long**) corto (**short**) con signo (**signed**) sin signo (**unsigned**)

Variables
Es un identificador que sirve para almacenar valores de un tipo de dato, se declara:
Tipo de dato x,y,z;
en donde: x,y,z son los nombres de variables
Tipo de dato: es un tipo int, float, double, ...
Ejemplo: int x, y, z; float raiz,area;

Sentencias (instrucciones, ordenes, …)
Sentencias de asignación
1. Asignación directa con el signo de igualdad (previo a la asignación debe declararse las variables):
variable = valor asignado
En donde 'valor asignado' puede ser una constante una variable o una expresión.
El 'valor asignado' se convierte al tipo de dato de la 'variable' si es válido en la declaración.
Ejemplos:
letra=A; n=2*3; i=n+1; i=i+1;
raiz = 4+cuadrado (3); //cuadrado es una función.
m=4*2.2; // si m es de tipo int 'm' almacena 8
//En todos los ejemplos, debe haberse declarado las variables

2. Durante la entrada de datos (al digitar valores usando teclado):
Ejemplo:
scanf("especificación de tipo",dirección de variable)
scanf: función de biblioteca que permite leer datos desde el teclado y asignarlos a las variables.
Algunas especificaciones de tipo:
%f para tipo float %d para tipo int
%c para tipo char %s para tipo cadena de caracteres
Adicional investigar: %ld %lf
Se coloca el nombre de la variable anteponiendo el operador &:
Ejemplos:
scanf("%d",&n); scanf("%f",&raiz); scanf("%c",&letra); scanf("%s",mensaje);

Operadores
Operadores aritméticos
Con los datos constantes o variables de los tipos numéricos mencionados se pueden realizar las siguientes operaciones, en orden de jerarquía descendente:

Operación	**Símbolo**	**Expresión**	**Resultado**
multiplicación	*	x*y	producto de x por y
división	/	x/y	cociente de x entre y
modulo	%	x%y	resto de la división de x entre y (x e y enteros)
adición	+	x+y	suma de x e y
sustracción	-	x-y	diferencia de x e y

Verificar cálculos para:
1. Del mismo tipo de datos de los datos a operar
2. Del tipo de mayor rango si los datos son de tipo distintos.

Se pueden utilizar los siguientes caracteres, siendo el orden de precedencia de mayor a menor:

a)Paréntesis b)*,/,% y c)+,-
Las operaciones que tienen igual precedencia se procesa de izquierda a derecha. Ejemplos (verificar):
15/2+3.0*2 es igual a 7+6.0 = 13.0 (punto flotante)
15/2.0 – 3.0*2 es igual a 7.5+6.0 = 13.5
(3-4.7)*5 es igual a –1.7*5 = -8.5
1+45%12 es igual a 1+3 = 4 (entero)

Operadores relaciónales:

==	igual	>	mayor
!=	no igual	<	menor
&&	Y lógica	>=	mayor o igual que
||	O lógica	<=	menor o igual que
?:	condicional		

Operadores monarios :

&	Dirección	*	Puntero
++	Incremento	--	Decremento

Operadores de asignación :

=	Asignación simple	+=	Asignación suma
-=	Asignación resta	*=	Asignación multiplicación
/=	Asignación división	%=	Asignación modulo

Expresiones
Las expresiones en general son derivadas del uso de las variables, por ejemplo:
X=(A+(B+C)*(B+C))/D;
En general las expresiones algebraicas, se pueden expresar en lenguaje C, y se pueden utilizar funciones de biblioteca y/o de usuario. En el siguiente ejemplo se utiliza la función de biblioteca del lenguaje “sqrt”.
X=-B+sqrt((B*B)-(4*A*C))/(2*A);

Otras expresiones en c

1.
```
/*  0 = masculino, si no femenino, y los cuenta */
if (x==0) { printf(“masculino”); masculino = masculino + 1; }
else { printf(“femenino”); femenino = femenino + 1; }
```

2.
```
/*  0 = masculino, si no femenino, y los cuenta */
if (x==0) { printf(“masculino”); ++masculino; }
else { printf(“femenino”); ++femenino; }
x = ++masculino;
```

3. Ej. “masculino = 99, se incrementa a 100, 100 se almacena en x”:
```
x = masculino++;        //pos incremental
```

Ej. “masculino = 99, se almacena en x, y se incrementa a 100”:
De igual manera para masculino - -

Entradas y salidas básicas
Las entradas de datos básicas, por defecto se realizan por el teclado y las salidas de datos por la pantalla.
Para la entrada de datos se utilizará: scanf(“%f”,&precio);
Se asignará en la variable precio el valor digitado por el teclado.
Y para la salida de datos se utilizará: printf(“%f”,precio);
Se imprimirá en la pantalla el valor que almacene la variable precio.

SEGUNDA SEMANA:

Estructura secuencial
Aquellas que ejecutan sentencias secuencialmente. Por ejemplo :

```
int a,b;  a=10;  b=11; printf(“%d”,a+b);   // se ejecutarán secuencialmente hasta imprimir el valor de a+b.
```

Estructura selectiva
Aquellas que ejecutan sentencias según una condición, por ejemplo :

.Sentencia if
La instrucción if sirve para ejecutar código sólo si una condición es cierta.

```
if ( condición )  instruccion1(proposición);
```

```
if ( condición )  instruccion1(proposición);
   else instruccion2(proposición);
```

Si condición es verdadero ejecuta instruccion1 en caso contrario instruccion2 (opcional)

Ejemplo :

```
if (a>b) max=a;
else      max=b;

if  ( condición )
          sentencia
```

Ejemplo:

```
int x = 1;
main()
{
  if ( x == 1 )       printf (“la variable x vale uno\n”);
  if ( x>=2 && x<=10 )     printf (“x está entre 2 y 10\n”);
}
```

Usando Bloques de instrucciones ó sentencias { } :

```
int x = 1;
main()
{
if ( x == 1 )
   {
    printf (“la variable x vale uno\n”);
    .....
    .....
   }
if ( x>=2 && x<=10 )
   {    printf (“x está entre 2 y 10\n”);    }
}
```

.Sentencia if else

La construcción if else sirve para ejecutar código sólo si una condición es cierta ó en caso contrario ejecutar código si es falsa, la sintaxis es:

```
if  ( condición )
      sentencia
else
     sentencia
```

Ejemplos:

```
....

if (a>b) max=a;

else max=b;

....
```

También podemos usar como se muestra ó con { } :

```
....
....

if (a>b)
           if (c>d) max=c;
else
          if (e>f) max=e;
....
....
```

.Sentencia if else if

....

....

```
 if ( a == b )
       {
          puts(“ Los numeros son iguales\n");
       }
          else if ( a  > b )
             {
                 printf("%d   es mayor a : b\n", a);
              }
```

.Sentencia switch
Ejecuta acciones diferentes según el valor de una expresión:

```
switch ( expresión )
{
          case valor1:
            ... sentenciasA...
               break;
          case valor2:
            ... sentenciasB ...
               break;
          case valor3:
          case valor4:
            ... sentenciasC ...
          default:
            ... sentenciasD ...
}
```

Ejemplo:

```
.....
int opcion;  printf("Escriba 1 si desea continuar; 2 si desea terminar: " );  scanf ( "%d", &opcion );
switch ( opcion )
{
             case 1:
                    printf (“Continua \n");   break;
             case 2:
                    salir = 1;  printf (“Salir \n");    break;
             default:
                    printf (“Opción no reconocida \n");
}
```

Ejemplo:

```
.....
switch(error)

{
  case 0:printf("\r\nError de sintaxis\r\n");break;
  case 1:printf("Falta paréntesis \r\n");break;
  case 2:printf("Falta paréntesis \r\n");break;
  case 3:printf("Demasiados paréntesis\r\n");break;
  case 4:printf(“Error de paréntesis 1 \r\n");break;
  case 5:printf(“Error de paréntesis 2");break;
  case 6:printf("Error en la posición paréntesis\r\n");break;
  case 7:printf("Numero ilegal de argumentos");break;
  case 8:printf("Carácter ilegal");break;
  case 9:printf(“Digito hexadecimal despues de $");break;
  case -1:busqueda(codigo); //no hay error
}
```

Estructuras iterativas
Aquellas que ejecutan sentencias repetidamente controlada por una condición (bucle). Por ejemplo:

for
Sintaxis:

```
for ( expresión_inicial; condición; expresión_de_paso )
             sentencia(s)
```

La expresión_inicial se ejecuta antes de entrar en el bucle.
Si la condición es cierta, se ejecuta sentencia(s) y después expresión_de_paso.
Luego se vuelve a evaluar la condición, y así se ejecuta la sentencia(s) una y otra vez hasta que la condición sea falsa. Ejemplo:

Ejemplo típico de uso:

```
int i;
...
for (i=0; i<10; i++)
{      printf (“%d\n“,i);  }
.....
.....
```

while

```
int a=10,b=0;
while ( a>b )
{     b=b+1; printf(“%d\n”,b); }
```

do while //(opcional)

```
#include <stdio.h>
#include <string.h>
main ()
{
  char dato[80] = “uni";  char password[80]  = "";
  do {    printf ("Digite contraseña:\n");        scanf("%s", password);    }
  while (strcmp(password, dato));
  printf (“password correcto.\n");
}
```

Bucle infinito

```
{
.....
          i = 0
          while ( i == 0 )    {   j++;     }
.....
.....

}
```

Otra sentencia: break, permite salir de una estructura repetitiva.

TERCERA SEMANA:

Array, es una variable, determinada como un conjunto de elementos del mismo tipo de datos ordenados e identificados por índices para cada elemento, en cada elemento se almacena un dato.

Arrays Unidimensionales

```
int b[10];              // declara una matriz de 10 elementos enteros, matriz de una dimensión.
float c[10];            // declara una matriz de 10 elementos numéricos reales, matriz de una dimensión.

int a[n];         // declara una matriz de “n” elementos enteros, “n” debe tener asignado un valor valido.
                  // “n” es el tamaño en la primera dimension
```

a[0], a[1], a[2], a[3], a[4], a[5], a[6], a[7], a[8] y a[9] … a[n-1] “n” Elementos es de 0 a n-1

En principios todas las matrices son de una dimensión, la dimensión principal.

Inicialización de arrays

En ocasiones la declaración puede incluir una inicialización de la matriz como en los siguientes ejemplos:

```
const int an[5] = {1, 2, 3, 4, 5};
int ax[6] = {1,2,3,4};                  // == {1,2,3,4,0,0}
int ai[ ] = {1,2,3,4,5};                //
```

Cuando el tamaño señalado es menor que la lista de inicio se produce un error.
Si el tamaño es mayor, se rellenan los espacios sobrantes con ceros (para numéricos).

En general las matrices se declaran directamente mediante una expresión con el tipo de dato:
int m1[10]; // matriz de 10 enteros
int *m4[10]; // matriz de 10 punteros a entero

Procesamiento de arrays
Los array se trata en general como una variable que almacena en cada elemento un dato.

Compruebe: Ordenación de datos de un array (arreglo-matriz)

```
#include "stdio.h"
main()
{
        float dato[50],temporal; int k,i,j;
        printf("ingrese la cantidad de datos :"); scanf("%d",&k);  //no validar dato
        // lectura de datos
        printf("ingrese %d valores \n",k);
        for (i=0;i<k;i++) scanf("%f",&dato[i]);
        // ordenacion de datos ingresados
        for (i=0;i<k;i++)
          for (j=i+1;j<k;j++)
             if(dato[j]<dato[i])
               {temporal=dato[i]; dato[i]=dato[j]; dato[j]=temporal;}
        // imprimir resultados
        printf("datos ordenados \n");
        for (i=0;i<k;i++)
          printf("%10.3f",dato[i]);   // 3=tres dígitos como decimales, 10=cantidad de dígitos de cada dato[i]
}
```

Arrays bidimensionales
tipoX m[a]; //Matriz unidimensional tipoX m[a][b]; //Matriz bidimensional
tipoX m[a][b][c]; //Matriz tridimensional

```
int dias[2][12] = {
  {31,28,31,30,31,30,31,31,30,31,30,31},
  {31,29,31,30,31,30,31,31,30,31,30,31} };
....
```

int Tabla[10][10]; char DimensionN[4][15][6][8][11]; //verificar
Cada elemento de Tabla, desde Tabla[0][0] hasta Tabla[9][9] es un entero. Del mismo modo, cada elemento de Dimension N es un carácter.
DimensionN[3][11][0][4][6] = DimensionN[0][12][5][3][1]; Tabla[0][0] = Tabla[0][0]+Tabla[9][9];

Inicialización

```
int i,j,x[3][3];    //analizar
for (i=0;i<3;i++)
{
  for (j=0;j<3;j++)
  {
        x[i][j]=0;
        printf("Inicializando variable x[%d][%d]=%d \n",i,k,j,x[i][j]);
  }
}
```

Arrays multidimensionales
Un array multidimensional es un array de tres ó más dimensiones. Si tenemos un array de N dimensiones, cada dimensión de tamaño d1,d2,..,dN, el número de elementos del array será d1*d2*..*dN, y para acceder a un elemento concreto del array utilizaremos N índices, cada uno de los cuales referenciará a una posición dentro de una dimensión, siempre según el orden de declaración.

```
int precios[10][20][30]; int i,j,k,dato=0,x[3][3][3];
for (i=0;i<3;i++)
{  for (j=0;j<3;j++)
   {
      for (k=0;k<3;k++)
```

```
    {    x[i][j][k]=dato; printf("Inicializando variable x[%d][%d][%d]=%d\n",i,k,j,x[i][j][k]);  }
  }
}
```

CUARTA Y QUINTA SEMANA:

Funciones de biblioteca ó estándar: Ejemplo algunas funciones de cadenas

```
#include "stdio.h"
#include "conio.h"
#include <ctype.h>
#include <string.h>
int F_1(int x[]);
main()
{   int E,P,A,Dpto,SE,i,j,k,l,m; int x[2]; char NOMBRE[50]="UNI-FIIS"; x[0]=11; x[1]=21;
    E=F_1(x);  printf("%d\n",E); printf("%s\n",NOMBRE);     printf("INGRESE NOMBRE\n");
    scanf("%2s",&NOMBRE);    printf("%2s\n",NOMBRE);
    char car = 'X';
    if (isalnum(car)) { printf("%c es alfanumerico\n",car); }
      else printf("%c no es alfanumerico\n",car);
    if (isalpha(car)) { printf("%c es alfabetico\n",car); }
      else printf("%c no es alfabetico\n",car);
    if (isdigit(car)) { printf("%c es un digito\n",car); }
      else printf("%c no es un digito\n",car);
    if (isspace(car)) { printf("%c es un espacio en blanco\n",car); }
      else printf("%c no es un espacio en blanco\n",car);
    if (isupper(car)) { printf("%c es mayuscula\n",car); }
      else printf("%c no es mayuscula\n",car);
    if (islower(car)) { printf("%c es minuscula\n",car); }
      else printf("%c no es minuscula\n",car);
    char wvar1[50]="121"; char wvar2[50]="120"; int ptr; ptr = strcmp(wvar1, wvar2);
    if (ptr > 0) { printf("1 es mayor que 2\n"); }
    if (ptr < 0) { printf("1 es menor que 2\n");}
    if (ptr == 0) {printf("2 es igual a 1\n"); }
    int c; c=getch();
}
int F_1(int x[]) { int z; z=x[0]+x[1]; return z; }
```

Otras funciones de biblioteca

1 abs(i) int stdlib.h
int abs(int i); Devuelve el valor absoluto de i x = abs(-7) // x es 7
2 acos(d) double math.h
double acos(double d); Devuelve el arco coseno de d angulo = acos(0.5); // angulo devuelto es phi/3
3 asin(d) double math.h
double asin(double d); Devuelve el arco seno de d angulo = asin(0.707); // aproximadamente phi/4
4 atan(d) double math.h
double atan(double d);
long double tanl(long double d); Devuelve la arco tangente de d. Calcula el arco tangente del argumento x. Requiere el llamado de la biblioteca complex.h angulo atan(1.0); // angulo es phi/4
5 atan(d1, d2) double math.h
double atan(double d1, double d2); Devuelve el arco tangente de d1/d2 angulo = atan(y, x)
6 atof(s) double stdlib.h
double atof(const char *cadena) Convierte la cadena s a una cantidad de doble precisión. Requiere el llamdo de la biblioteca math.h double x;

```
char *cad_dbl = "200.85"; ...   x=atof(cad_dbl); // convierte la cadena "200.85" a valor real
```

7 atoi(s) int stdlib.h
int atoi(const char *cadena) Convierte la cadena s a un entero.
La cadena debe tener el siguiente formato :
[espacio en blanco][signo][ddd] (siendo obligatorio los digitos decimales). int i;

```
char *cad_ent="123";
...
i =atoi(cad_ent);  //convierte la cadena "123" al entero 123
.... Otros ..
.........
```

8 fclose(f) int stdio.h
int fclose(FILE *f); Cierra el archivo f. Devuelve el valor 0 si el archivo se ha cerrado con exito. int fclose(FILE "archivo");

9 feof(f) int stdio.h
int feof(FILE *f); Determina si se ha encontrado un fin de archivo. si es asi, devuelve un valor distinto de cero, en otro caso devuelve 0 feof(fichen);
10 fgetc(f) int stdio.h
int fgetc(FILE f); Lee un caracter del archivo f c+fgetc(fp)
11 fegts(s, i, f) char(puntero) stdio.h
char *fgets(char s, int s, FILE *f); Lee una cadena s, con i caracteres, del archivo f fgets(caddemo, 80, fp);
12 floor(d) double math.h
double floor(double d); Devuelve un valor redondeado por defecto al entero menor mas cercano x=floor(6.25); // x vale 6
13 fmod(d1, d2) double math.h
double fmod(double d1, double d2); Devuelve el resto de d1/d2 (con el mismo signo que d1)
resto=fmod(5.0,2.0); // resto igual a 1.0
14 fopen(s1, s2) file(puntero) stdio.h
FILE *fopen(const char *s1, const char *s2) Abre un archivo llamado s1, del tipo s2. Devuelve un puntero al archivo. *
Modo Accion

"r" Abre para lectura	"w" Abre un archivo vacio para escritura
"a" Abre para escritura al final del archivo	"r+" Abre para lectura/escritura
"w+" Abre un archivo vacio para lectura/escritura	"a+" Abre para lectura y anadir
"rb" Abre un archivo binario para lectura.	"wb" Abre un archivo binario para escritura
"ab" Abre un archivo binario para anadir	"rb+" Abre un archivo binario para lectura/escritura.
"wb+" Abre un archivo binario para lectura/escritura	"ab+" Abre o crea un archivo binario para lectura/escritura

if ((corriente2=fopen("datos","w+"))==NULL {printf("El archivo...no se ha abierto \n"); }
15 fprintf(f, ...) int stdio.h
int fprintf(FILE *f, const char *formato [,arg,...]); Escribe datos en el archivo f (el resto de los argumentos fprintf(f1, "El resultado es %f\n",result);
17 fputc(c, f) int stdio.h
int fputc(int c, FILE *f); Escribe un caracter en el archivo f fputc(*(p++), stdout);
18 fputs(s, f) int stdio.h
int fputs(const char *cad, FILE *f) Escribe una cadena de caracteres en el archivo f fputs("es una prueba", f1);
19 fread(s, i1, i2, f) int stdio.h
size_t fread(void *b, size_t t, size_t n, FILE *f); Lee i2 elementos, cada uno de tamano i1 bytes, desde el archivo f hasta la cadena s fread(buf, strlen(msg)+1, 1, flujo);
20 fscanf(f, ...) int math.h
int fscanf(FILE *f, const char *formato, [, direccion,...]); Lee datos del archivo f (el resto de los argumentos fscanf(flujo, %s%f, cad, &f);
21 fseek(f, l, i) int stdio.h
int fseek(FILE *f, long desplaza, int origen); Mueve el puntero al archivo f una distancia de 1 bytes desde la posicion i (i puede representar el principio del archivo, la posicion actual del puntero o el fin del archivo.
Origen Significado
SEEK_SET Principio de archivo SEEK_CUR Posicion actual puntero SEEK_END Final del archivo
fseek(f1,OL,SEEK_SET); // ir al principio
22 ftell(f) long int stdio.h
long int ftell(FILE *f); Devuelve la posicion actual del puntero dentro del archivo f ftell(fichen)
23 fwrite(s, i1, i2, f) int stdio.h
size_t fwrite(const void *p, size_t i1, size_t i2, FILE *f); Escribe i2 elementos, cada uno de tamano 1 bytes, desde la cadena s hasta el archivo f
num=fwrite(lista,sizeof(char),25,flujo);
24 getc(f) int stdio.h
int getc(FILE *f); Lee un caracter del archivo f while(c=getc(fx) !=EOF {
print ("%c",c); }
25 getchar( ) int stdio.h
int getchar(void); Lee un caracter desde el dispostivo de entrada estándar int c;
while((*c=getchar()) != '\n')
print ("%c",c);
26 gets(s) char(puntero) stdio.h
char *gets(char *cad); Lee una cadena de caracteres desde el dispositivo de entrada estándar gets(nombre);
27 isalpha(c) int ctype.h
int isalpha(int c); Determina si el argumento es alfabetico. Devuelve un valor distinto de cero si es cierto; en otro caso devuelve 0. int c;
if (isalpha(c)) printf("%c es letra\n",c);
28 isascii(c) int ctype.h
int isascii(int c); Determina si el argumento es un caracter ASCII. Devuelve un valor disitinto de cero si es cierto; en otro caso devuelve 0 int c;
if (isascii(c)) printf('%c es un ascii\n",c)
29 iscntrl(c) int ctype.h

int isacntrl(int c); Determina si el argumento es un caracter ASCII de control. Devuelve un valor distinto de cero si es cierto; en otro caso devuelve 0 if(iscntrl(c)) printf"%c es un caracter de control\n",c);
30 isdigit(c) int ctype.h
int isdigit(int c); Determina si el numero es un digito decimal. Devuelve un valor disitinto de cero si es cierto; en otro caso devuelve 0 if(isdigit(c)) printf"%c es un digito\n",c);
31 isgraph(c) int ctype.h
int isgraph(int c); Determina si el argumento es un caracter ASCII grafico (hex 0x21 -0x7e; octal 041 -176). Devuelve un valor distinto de cero si es cierto; en otro caso devuelve 0 if(isgraph(c)) printf"%c es un caracter imprimible(no espacio)\n",c);
32 islower(c) int ctype.h
int islower(int c); Determina si el argumento es ua minuscula. Devuelve un valor distinto de cero si es cierto; en otro caso devuelve 0 if(islower(c)) printf"%c es una letra minuscula\n",c);
33 isodigit(c) int ctype.h
int isodigit(int c); Determina si el argumento es un digito octal. Devuelve un valor distinto de cero si es cierto; en otro caso devuelve 0 if(isodigit(c)) printf"%c es un digito octal\n",c);
34 isprint(c) int ctype.h
int isprintint c); Determina si el el argumento es un caracter ASCII imprimible (hex 0x20 -0x7e; octal 040 -176). Devuelve un valor distinto de cero si es cierto; en otro caso devuelve 0 if(isprint(c)) printf("\n"c imprimible\n",c);
35 ispunct(c) int ctype.h
int ispunct(int c); Determina si el argumento es un caracter de puntuacion. Devuelve un valor distinto de cero si es cierto; en otro caso devuelve 0 if(ispunct(c)) printf"%c es un caracter de puntuacion\n",c);
36 isspace(c) int ctype.h
int isspace(int c); Determina si el argumento es un espacio en blanco. Devuelve un valor distinto de cero si es cierto; en otro caso devuelve 0 if(isspace(c)) printf"%c es un espacio\n",c);
37 isupper(c) int ctype.h
int isupper(int c); Determina si el argumento es una mayuscula. Devuelve un valor distinto de cero si es cierto; en otro caso devuelve 0 if(isupper(c)) printf"%c es una mayuscula\n",c);
38 isxdigit(c) int ctype.h
int isxdigit(int c); Determina si el argumento es un digito hexadecimal. Devuelve un valor distinto de cero si es cierto; en otro caso devuelve 0 ifisxdigit(c)) print"%c es un digito hexadecimal\n",c)

Otros prototipos de funciones de biblioteca del lenguaje
Estas son algunas de las funciones:
alloc.h : Existen funciones para asignar, liberar memoria, u obtener informacion de bloques de memoria.
ctype.h : Son funciones que nos permiten conocer la naturaleza de un caracter, o bien para convertir de mayusculas a minusculas y viceversa; y valores enteros a codigos ASCII.
dir.h : Permite ordenar, crear, modificar, mover y eliminar directorios
errno.h : Representa los numeros de error, despues que ocurre un error se puede consultar el valor de la variable del sistema de errno para obtener mas información sobre ese error.
float.h : Define los limites de los tipos de coma flotante
limits.h : Define los limites de los diferentes tipos de enteros
math.h : Contiene las funciones matematicas estándar utilizadas en C y C++
setjmp.h : Define el tipo de jmp_buf para algunas funciones.
signal.h : Contiene funciones de estado.
stdarg.h : Define funciones que pueden ser llamadas con diferentes numeros de argumentos, de modo que se pueda escribir f(a) y f(a,b).
stdef.h Se definen algunos tipos especiales
stdio.h : Incorporan las funciones de Entrada - Salida E/S estándar, tipos y macros
stlib.h : Declara funciones que son utiles para diferentes propositos, en especial de busqueda y ordenacion.
string.h : Este archivo contiene funciones para manejo de cadenas de caracteres.
time.h : Contiene funciones relativas a fechas y horas

Funciones de usuario
Declaración de Funciones de usuario
TipodeDatoqueDevuelve nombrefunción (lista de parámetros/argumentos)
{ cuerpo de la función }

El “TipodeDatoDevuelve” y la “lista de parámetros/argumentos” (puede ser vacia: void) contiene una serie de identificadores separados por comas si hay mas de uno, a través de los cuales se espera pasar valores a la función.
La declaración de los tipos de los parámetros sirve para especificar los tipos de parámetros.
El cuerpo de la función se compone de instrucciones como en la función main, es decir puede contener sus propias variables (locales a la función), bloques, instrucciones de entrada y salida, de control, llamadas a otras funciones, etc. Ejemplos de declaraciones:
void F_MENSAJE(char msg[]); void F_CALCULAR(); void F_X(int a, float b);

Definición de funciones de usuario
Una función es simplemente un grupo de instrucciones que se ejecutan cada vez que se invoca el nombre de la función, posiblemente con algunos valores, llamados argumentos. Cuando termina la ejecución se retorna al punto de llamada.

Sentencia return
Una función se ejecuta hasta que se encuentre el final del cuerpo de la función ó hasta que se encuentre una función de retorno:

```
return;
return ( expresión );
```

En ambos casos finaliza la ejecución de la función, pero en el segundo además devuelve el valor de la expresión.

Función main
Es la función principal de todo la estructura de un programa en c/c++, tiene todas las características de una función definida.
Ejemplos:

```
void main(void)        int main(void)        char main(int a, char b)
```

Valores devueltos por las Funciones
La función principal como las funciones en general pueden devolver valores de determinados tipos de datos que se definen así:
Para la función principal:

```
void main(void)// devuelve vacio (no devuelve “nada”)
int main(void)// devuelve un valor entero de tipo int
char main(int a, char b)// devuelve un valor de un carácter ASCII de tipo char
float F_TOTAL(int proceso);// devuelve un valor real de tipo float
```

Argumentos de las funciones
Son identificadores/variables utilizados para recibir valores, para ser procesados por la función.

```
void f_mensaje(char msg[]);              // recibe un valor de tipo char en la variable msg[]
void f_fecha();                          // no recibe valores
void f_hora();                           // no recibe valores
float f_totalfactura(int nrofactura)     // recibe un valor de tipo int en la variable nrofactura
float f_promedio(int codigo)             // recibe un valor de tipo int en la variable codigo
```

Paso de argumentos por valor
Se utilizan las funciones por llamada por valor de la siguiente manera:

```
....
f_mensaje(" hola");
....

float promedio;          promedio = f_promedio(991225A);
....

float totalfactura;      totalfactura =  f_totalfactura(1009856)
....
```

Paso de argumentos referencia
Se utilizan las funciones por llamada por referencia de la siguiente manera:

```
....
float radio,area;    radio = 5;   area=f_areacirculo(radio);    printf("Area : %f",area);
....

// Ejemplo : funciones ( F_ )
#include "stdio.h"
#include <string.h>
#include <dos.h>
void f_mensaje(char msg[]);  void f_fecha();  void f_hora();  float f_areacirculo(float radio);
main()
{
  f_mensaje(" hola ");    printf("\n");    f_fecha();    printf("\n");    f_hora();
  printf("\n");    float radio,area;    radio = 5;    area=f_areacirculo(radio);  // por referencia
  printf("Area : %f",area);
}
```

```
// Envia mensaje en la pantalla
void f_mensaje(char msg[])
{   int i,longitud;     longitud =strlen(msg);    printf("Mensaje : ");
    for ( i=0; i< longitud ; i++ )       {  printf( "%c ", msg[ i ] ) ;  }
}

// Obtiene la fecha del sistema
void f_fecha(void)
{   struct date d;    getdate(&d);    printf("%d/%d/%d",d.da_day,d.da_mon,d.da_year);  }

// obtiene la hora del sistema
void f_hora(void)
{   struct  time t;    gettime(&t);    printf("%d:%d:%d",t.ti_hour,t.ti_min,t.ti_sec);  }

// obtiene el área de un círculo
float f_areacirculo(float radio)
{   float area;        area = radio*radio*3.1415;          return area;  }
```

Arrays como argumentos (paso de arrays a funciones)

```
#include "stdio.h"
#include "conio.h"
int F_ARRAY_SUMA(int a[ ],int n);
main()
{
  int d;  printf("Cuantos datos va a sumar \n");   scanf("%d",&d);     int x[500];
  for(int i=0;i<d;i=i+1)   {   printf("Ingrese dato numero %d\n",i+1);   scanf("%d",&x[i]);     }
  printf("La suma es=%d\n",F_ARRAY_SUMA(x,d));  getch();
}
int F_ARRAY_SUMA(int a[],int n)
{ int s=0;
  for(int i=0;i<n;i=i+1)   {    s=s+a[i];  }  return s;
}
```

El paso de una matriz en su conjunto (multidimensional o no) como argumento de una función es diferente al paso de un elemento de una matriz. También se enuncia argumento “array en funciones”. Ejemplo:

```
....
int dias[2][12] = {
   {31,28,31,30,31,30,31,31,30,31,30,31},
   {31,29,31,30,31,30,31,31,30,31,30,31} };
....
x = F_fun1(dias, f, c, d, e);     // paso de matriz
z = F_fun2(dias[f][c], d, e);    // paso de elemento
```

Recursividad

Se permite hacer llamadas recursivas: Se conocen tambien como funciones anidadas, en el cuerpo de la función se invoca a si misma.

Ejemplo:

```
float F_factorial (int n)
{   if (n<=1)    {  return 1.0;   }
    else   {      return n*F_factorial(n-1);     }
}
```

Verifique: no se pueden declarar funciones dentro de otras. Todas las funciones son globales.

SEXTA y SEPTIMA SEMANA:

Archivos

Un archivo es una colección de datos, que con un nombre se graba en un medio de almacenamiento: diskettes, disco duro, etc. Estos datos de un archivo pueden ser leidos, modificados, borrados. Los datos de un archivo son simplemente bytes o caracteres almacenados uno a continuación de otro, según el formato que se utiliza para grabarlos. Las Bases de Datos (“archivos de alto nivel”) se graban en un formato especial, y que son manipulados mediante el “lenguaje SQL (Structured Query Language - Lenguaje de Consulta Estructurada)”. Los conceptos de los llamados “archivos de acceso de bajo nivel”, son los que se utilizaran en el curso para comprender su uso y de las bases de datos NOSQL.

Es necesario abrir un archivo antes de escribir o leer. Despues de cada operación de escritura o lectura, el archivo apunta al siguiente componente desplazamiento secuencial (en general fila y/o registro del archivo). Es posible desplazar o ubicar el apuntador a un componente particular dentro del archivo, Empleando funciones

de acceso directo. Despues de realizar las operaciones de Lectura y Escritura se debe cerrar un archivo.

Entrada / Salida estandar

Se sabe que la entrada estandar de datos se realiza por el teclado, existen otros dispositivos de entrada de datos como Codigo de barras, Moden, un archivo, etc. La Salida Estandar de datos se realiza por la pantalla del Computador, existen otros dispositivos de salida como la Impresora, Moden, Plotter, un archivo, etc.
La entrada de datos por archivos se manejara con la función fscanf y la salida estandar a un archvio con la función fprintf, como se ve a continuación en los siguientes ejemplos.

Funciones de manejo de Archivos

Uso de fopen

Las siguientes 2 lineas, abre un archivo con acceso dependiendo del modo de acceso.
FILE *f1; *f1=fopen(“nombre del archivo”,modo);
donde modo : “w” para escribir “r” para leer “a” para añadir existen otros tipos
f1 es una variable de tipo puntero a archivo

Uso de fclose

fclose(f1); //cierra el archivo
Se usa despues de haber procesado un archivo y ya no necesitarlo, f1 es un puntero a archivo.

Uso de fprintf

Nos servira para escribir en archivos
fprintf(f1,“cadena de control de formatos de variables”,lista de variables)
Siilar al uso de printf, escribe datos con formato, pero la salida se realiza al archivo al que apunta f1.

```
fprintf(f1,"%1d %8s%1d %5.2f %-2d\n",art.codigo,art.nombre,art.codigo,art.precio,art.cantidad);
```

El formato significa que la variable art.codigo utilizara solo un digito, luego se escribe espacio en blanco, luego art.nombre utilizara 8 caracteres, luego art.codigo un solo digito, luego se escribe espacio en blanco, art.precio se escribe un numero real de 5 digitos con 2 decimales, luego se escribe espacio en blanco, art.cantidad se escribe justificado a la izquierda (signo menos) utilizando 2 digitos, y luego “salta” a la siguiente linea.
La estructura del la fila=registro del archivo esta definido por el formato de la función fprintf.

Uso de fscanf

Nos permitira leer archivos
fscanf(f1,“cadena de control de formatos de variables”,lista de variables)
Igual que scanf pero la lectura se hace en f1
El formato para la lectura debe ser coherente con el del fprintf, es decir de acuerdo a la estructura que tenga los registros del archivo. Cada fscanf lee un registro.
Para leer en forma secuencial hasta el final el archivo utilizaremos:

```
…..
……
  while( fscanf(f1,"%1d %9s %5f %2d",&v1,&v2,&v3,&v4)>0 )
  {    printf("%1d\n",v1);          printf("%9s\n",v2);          printf("%5f\n",v3);          printf("%2d\n",v4);
       fprintf(f2,"%1d %9s %5.2f %-2d\n",v1,v2,v3,v4);   suma=suma+v3*v4;   }
…..
…..
```

Mientras en el archivo existan registros la función (fscanf entre parentesis) “sera” mayor a cero.

Apertura y cierre de un archivo

Abre para solo lectura de un archivo existente: FILE *f1; f1=fopen(“datos.txt”,”r”);
Cierra del archivo abierto en la anterior línea: fclose(f1);
Las siguientes lineas, muestran la apertura para escritura, si el archivo no existe, crea el archivo, ó si existe el archivo lo borra y lo crea: FILE *f1; f1=fopen(“datos.txt”,”w”)

```
……..
fclose(f1);  //cierra el archivo
```

Las siguientes lineas, muestran la apertura de un archivo exixtente, para agregar despues del ultimo registro del archivo: FILE *f1; f1=fopen(“datos.txt”,”a”);

```
……..
fclose(f1);  //cierra el archivo
```

Creación de un archivo: FILE *f1; f1=fopen(“datos.txt”,”w”)

Ejemplo:

```
#include "stdio.h“   // Escritura lectura
main()
{   FILE *f1;   int n;    f1=fopen("datos2.txt","w");
   for (n=1;n<10;n++)
```

```
  {  fprintf(f1,"%6d",n*n);    }
  fclose(f1);     printf("Datos del archivo datos2\n");   f1=fopen("datos2.txt","r");
  while(fscanf(f1,"%d",&n)>0)     {  printf("%6d\n",n);   }
  fclose(f1);
}
```

Procesamiento de archivos

El procesamiento esta relacionado en general con el tipo de lectura estandar: Secuencial ó Aleatorio.
Para fines de estudio del curso se utilizará el modo secuencial, como se muestra en el siguiente ejemplo:
En el siguiente ejemplo se utiliza una variable de tipo estructura, para los datos de articulos, se simula el ingreso para 10 articulos. Note que la cadena nombre lo manejamos con punteros.

```
#include "stdio.h"
#include "stdlib.h"
void main(void)
{ FILE *f1,*f2;  int n;    struct articulo { int codigo; char *nombre; float precio; int cantidad;} art;
  f1=fopen("\\a.txt","w");
  for (n=1;n<10;n++)
  {    art.codigo=n;              art.nombre="articulo";          art.precio=2*n;             art.cantidad=3*n;
       fprintf(f1,"%1d %8s%1d %5.2f %-2d\n",art.codigo,art.nombre,art.codigo,art.precio,art.cantidad);   }
  fclose(f1);
  f1=fopen("\\a.txt","r");          f2=fopen("\\b.txt","w");   char v2[9];      int v1,v4;      float v3,suma=0;
  while( fscanf(f1,"%1d %9s %5f %2d",&v1,&v2,&v3,&v4)>0 )
  {     printf("%1d\n",v1);          printf("%9s\n",v2);          printf("%5f\n",v3);          printf("%2d\n",v4);
        fprintf(f2,"%1d %9s %5.2f %-2d\n",v1,v2,v3,v4);   suma=suma+v3*v4;   }
  printf("%6.2f\n",suma);   fclose(f1);      fclose(f2);
}
```

Otro ejemplo: Note que la cadena nombre lo manejamos con un array, ingresando datos por teclado y grabando en un archivo.

```
#include "stdio.h"
#include "stdlib.h"
void main(void)
{ FILE *f1,*f2;    int n;   struct articulo { int codigo; char nombre[10]; float precio; int cantidad;} art;
  f1=fopen("datos.txt","w");
  for (n=1;n<10;n++)
  {  printf("codi.\n");        scanf("%d",&art.codigo);      printf("prec.\n");        scanf("%f",&art.precio);
     printf("cant.\n");        scanf("%d",&art.cantidad);   printf("nomb.\n");      scanf("%10s",&art.nombre);
     fprintf(f1,"%1d %10s%d %5.2f %-2d\n",art.codigo,art.nombre,art.codigo,art.precio,art.cantidad);
  }
  fclose(f1);
}
```

Otro Ejemplo: Trabajando con varios archivos y buscando un registro existente.

```
#include "stdio.h"
void main(void)
{ FILE *f1,*f2;   int n;   struct articulo { int codigo; char *nombre; float precio; int cantidad;} art;
  f1=fopen("datos1.dat","w");    //buscar el archivo con el buscador de windows
  for (n=1;n<100;n++)
  { art.codigo=n;        art.nombre="articulo";    art.precio=2*n;     art.cantidad=3*n;
     fprintf(f1,"%05d %-s%d %10.2f %-2d\n",art.codigo,art.nombre,art.codigo,art.precio,art.cantidad);
  }
  fclose(f1);
  f1=fopen("datos1.dat","r");      f2=fopen("datos2.dat","w");
  int v1,v4;    char v2[9];      float v3;    int busca;     busca = 5;   //  scanf("%1d",&busca);
  while( fscanf(f1,"%5d %9s %5f %2d",&v1,&v2,&v3,&v4)>0 )
  {   if (v1==busca)  {  printf("%2d\n",v4);   fprintf(f2,"%1d %9s %5.2f %-2d\n",v1,v2,v3,v4);   }   fclose(f1);
  fclose(f2);
}
```

OCTAVA SEMANA: **EXAMEN PARCIAL**

NOVENA SEMANA:

Clases de objetos, objetos y atributos

Una clase o clase de objetos es una abstracción que describe un grupo de objetos-instancias con propiedades

(atributos) comunes y comportamiento (metodos-operaciones) comunes y (lo que es más importante) una semántica común. Ejemplos de clases de objetos: Muebles, Documentos comerciales, Clientes, Vectores, figuras geometricas, etc.

Objeto: un objeto es algo abstracto o algo material, con unos límites definidos y que es relevante para un problema en cuestión. Es una instancia (irrepetible) que pertenece a una clase de objetos.
Los modelos de objetos sirven tanto para obtener un conocimiento mejor del dominio del problema. Ejemplos : La mesa del comedor de la FIIS, la factura Nro.1962 de la empresa x, El Cliente con RUC 10020030011, el vector (1,10)
El término objeto en la vida diaria se utiliza para referirnos tanto a clases de objetos (por ejemplo el concepto abstracto mesa) como a las instancias de estas clases (una mesa determinada). Es mejor utilizar los términos clase y objeto-instancia para evitar confusiones.

Atributo: es una caracteristica, es un dato contenido en todos los objetos-instancias de una clase. Cada atributo tiene un valor para cada una de los objetos-instancias. Varias clases pueden tener atributos comunes (por ejemplo nombre, en las clases Persona y Calle) pero cada atributo debe ser único dentro de una clase.

Los atributos tienen que ser datos, no objetos. La diferencia entre unos y otros reside en la identidad: los objetos tienen identidad, pero los atributos no. Ejemplo : El código de un alumno, el RUC de un cliente, las coordenadas de un vector. Los atributos se representan en la segunda área de los símbolos de clase e instancia. En las clases, figurará el nombre del atributo, el tipo y el valor por defecto. En las instancias, el valor del atributo para ese objeto determinado.

DECIMA SEMANA:

Encapsulamiento
La esencia del encapsulamiento ó encapsulación se refiere cuando un objeto desde el punto vista funcional oculta la complejidad de los procedimientos. Ejemplo: Al ver la televisión, no vemos la complejidad que existe detrás de la pantalla.

Metodo
Un método es una operación, lleva implícito un objeto destino, sobre el que se va a realizar la operación. El comportamiento de la operación depende de la clase del objeto destino. Todos los objetos de una clase comparten las mismas operaciones o métodos. Ejemplo: Visualizar formulario, Imprimir factura, sumar vectores, imprimir coordenadas de vector. Los Mensajes son “las formas en que se comunican los objetos.

```
#include "stdio.h"
#include "stdlib.h"
class articulos
{ public : int k_art; char n_art[20];    void agregar(void);    void buscar(void);  };
void articulos::agregar(void)
{  FILE *f1;    f1=fopen("articulos.dat","w");    fprintf(f1,"%5d %20s\n",k_art,n_art);   fclose(f1);  }
void articulos::buscar(void)
{ int wk_art; char wn_art[20];   FILE *f1;    f1=fopen("articulos.dat","r+");
  while ( fscanf(f1,"%5d %20s\n",&wk_art,&wn_art)>0 )
  {  if ( wk_art==k_art )    {    printf("Dato lujan encontrado: %10s\n",wn_art);     }   }
  fclose(f1);
}
void main(void)
{ articulos x1;  printf("Datos a agregar\n");   scanf("%5d",&x1.k_art);    scanf("%20s",&x1.n_art);
  x1.agregar();   printf("Datos a buscar\n");   x1.k_art=3;   scanf("%5d",&x1.k_art);   x1.buscar();
}
```

DECIMA PRIMERA SEMANA:

Constructor
El “constructor” es un método cuyo bloque de código que se ejecuta al crear una variable objeto de una clase.

Destructor
El “destructor” es un bloque de código que se ejecuta cuando la variable objeto se “elimina”.

```
#include "stdio.h"   //analizar
#include "stdlib.h"
class articulos
{
  public : int k_art; char n_art[20];    public : articulos(){k_art=99;};
```

```
  public : ~articulos(){printf("ejecutando destructor de la clase");};
  void agregar(void);   void buscar(void);
};
void articulos::agregar(void)
{  FILE *f1;   f1=fopen("\\articulo.txt","a");    fprintf(f1,"%5d %20s\n",k_art,n_art);    fclose(f1);  }
void articulos::buscar(void)
{ int wk_art; char wn_art[20];   FILE *f1;  f1=fopen("\\articulo.txt","r");
  while ( fscanf(f1,"%5d %20s\n",&wk_art,&wn_art)>0 )
          {   if ( wk_art==k_art )   {   fprintf(f1,"1\n");   printf("Dato encontrado: %10s\n",wn_art);   }   }
  fclose(f1);
}
void main(void)
{ articulos x1;  printf("%d\n",x1.k_art);   printf("Agregar\n");   scanf("%5d",&x1.k_art);  scanf("%20s",&x1.n_art);
  x1.agregar();   x1.k_art=3;  printf("Buscar\n");   scanf("%5d",&x1.k_art);     x1.buscar(); }
```

DECIMA SEGUNDA SEMANA:

Herencia

```
#include "stdio.h"
#include "stdlib.h"
#include "conio.h"
class Barco
{   public:char *nombre;float peso;
};
class Carguero:public Barco
{  public:float carga;  };
class Acorazado:public Barco
{  public:int numeroArmas;int Soldados;  };
void main(void)
{  Barco x1;  x1.peso=10;   printf("%f\n",x1.peso);  Carguero x2;   x2.carga=200;   printf("%f\n",x2.carga);
   Acorazado x3;    x3.peso=300;   printf("%f\n",x3.peso);   getch();
}
```

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
class Persona
{   public:int dni;  void Hablar();  void Caminar();  };
class Empleado
{    public:float sueldo;    Persona jefe;  void Cobrar(); };
class empleado_tienda: public Persona, public Empleado
{   void AlmacenarStock();  void ComprobarExistencias();  };
main()
{    empleado_tienda x1;  x1.dni=123;  x1.sueldo=10.33;
     printf("%d   %f\n",x1.dni,x1.sueldo);  getch(); }
```

DECIMA TERCERA SEMANA:

Estructura

Una estructura es un tipo de dato que agrupa datos relacionados, definido por el usuario, para combinar varios tipos de datos previamente definidos, incluyendo otras estructuras previamente definidas. Ejemplo:

```
#include <stdio.h>
struct empleado {    int codigo; int  edad; int  calificacion;   } em1, em2;   struct empleado empleado99;
main()
{ em1.codigo = 10;                em1.edad = 15;              em1.calificacion = 75;
  em2.edad = em1.edad - 1;        em2.calificacion = 16;      em2.codigo = 11;
  printf("%d tiene %d años, cal. es de %d\n", em2.codigo, em2.edad, em2.calificacion);
  printf("%d tiene %d años y su cal. es de %d\n", em1.codigo, em1.edad, em1.calificacion);
}
```

Array de estructuras

```
#include <stdio.h>
struct emp {int codigo; int edad; int calificacion; } empleado[12];
main()
```

```
{ int i;
  for (i = 0; i < 12; i++)
  {  empleado[i].codigo = i;     empleado[i].edad = 16;  empleado[i].calificacion = 84;   }
  empleado[3].edad = empleado[5].edad = 17;  empleado[2].calificacion = empleado[6].calificacion = 92;
  empleado[4].calificacion = 57;    // empleado[10] = empleado[4];      /* solo en compiladores ANSI-C */
  for (i = 0; i < 12; i++)
  {  printf("%d tiene %d años y una calificacion de %d\n",empleado[i].codigo, empleado[i].edad,
                                                              empleado[i].calificacion);  }
}
```

Paso de estructuras a funciones de usuario

```
#include <stdio.h>
void F_prueba_struc(struct emp);  struct emp {int codigo; int edad; int calificacion;} empleado[12];
void main()
{ int i;
  for (i = 0; i < 12; i++)
  {    empleado[i].codigo = i;       empleado[i].edad = 16; empleado[i].calificacion = 84;  }
  empleado[3].edad = empleado[5].edad = 17;     empleado[2].calificacion = empleado[6].calificacion = 92;
  empleado[4].calificacion = 57;  empleado[10] = empleado[4];   F_prueba_struc(empleado[12]);
}
void F_prueba_struc(struct emp)
{  int i;
  for (i = 0; i < 12; i++)
  {  printf("%d tiene %d years y una calificacion de %d\n", empleado[i].codigo, empleado[i].edad,
                                                                  empleado[i].calificacion);   } }
```

DECIMA CUARTA SEMANA:

¿Qué es un PUNTERO?

El concepto de puntero, está relacionado con la dirección de memoria que "ocupa" un valor en la memoria de un CPU (Central Processing Unit - Unidad Central de Procesamiento), ya que denotan la dirección (address) o localización del valor de una variable puntero. La dirección de memoria determina dónde está almacenada. Un puntero es un variable que alacena una dirección de memoria y con esta "apunta" a otra variable/"objeto". No hay que confundir una dirección de memoria con el contenido de esa dirección de memoria.

int x = 1; // 1 es el valor,que alacena x, y x supongaos que tiene una dirección 1502.
int y = 2;

Dirección 1502 1504 1506 1508

		1	**2**				

La dirección de la variable x es 1502, se obtiene asi: &x
El contenido (valor) de la variable x es 1. El contenido(valor) de la variable y es 2

Un puntero tiene su **propia dirección** de memoria: &punt &car
Hay tantos tipos de punteros como tipos de datos.
Los punteros son importantes porque:
Permite que las funciones cambien el valor de sus argumentos.
Permite pasar vectores de forma eficiente entre funciones: en lugar de copiar cada elemento del vector, se copia la dirección del primer elemento.
Permite reservar memoria en tiempo de ejecución en lugar de en tiempo de compilación, lo que significa que el tamaño de un vector puede ser determinado por el usuario en lugar del desarrollador.

Operadores con punteros

Al trabajar con punteros se emplean dos operadores específicos:

- Operador de dirección: & representa la dirección de memoria de la variable que le sigue:
  &x representa la dirección de x
- Operador de contenido: *

El operador * aplicado al nombre de un puntero indica el valor de la variable apuntada:

```
int altura = 26.92,  *apunta;
apunta = &altura;     //inicialización del puntero
```

No se debe confundir el operador * en la declaración del puntero:

```
int x = 1, *p;
```

Con el operador * en las instrucciones:

```
p = &x;  *p = 100;     printf("\nContenido = %d", *p);
```

La variable p contiene la dirección de memoria de la variable x. La expresión: *p representa el valor de la variable (x) apuntada, es decir 1. La variable p también tiene su propia dirección: &p

Variables puntero
Un puntero guarda la dirección de un "objeto" en memoria.
Una variable puntero se declara como todas las variables. Debe ser del mismo tipo que la variable apuntada.
Su identificador va precedido de un asterisco *:

```
int *puntero1;  int *puntero2, *puntero3;   char variable, *punteroCaracter;    float *punteroReal, real;
```

En declaracion, declaramos un puntero a un entero. En la segunda, dos punteros a entero.
En la tercera, un carácter (variable simple) y un puntero a carácter (punteroCaracter).
Por último, punteroReal es un puntero a un real, y real es declarado como un número real.

```
#include <stdio.h>   // Compruebe
main()
{  int x = 1, y = 2; printf("Valores %d \t  %d \n",x,y);
   printf("Direcciones %d \t  %d \n",&x,&y);          int *p;  printf("Valor de p %d \t \n",p);  p = &x;
   printf("%d \t %d \n",&x,&y);      printf("Valor de p  &x=direccion %d \t \n",p);
   printf("Valor de *p despues de asignar &x=valor %d \t \n",*p);    /*1502   1504    1700
    x=1    y=2     p=1502  La variable p es declarado como un puntero a entero y se le asigna la direccion de x
    (&x) por lo que p se carga con el valor 1502  */
}
```

Expresiones de punteros
Si p y v son punteros:
Se puede sumar o restar valores enteros: p++, v+3, teniendo en cuenta que el desplazamiento (offset) depende del tipo de dato apuntado:

```
p++;              // p apunta a la siguiente dirección
v+=3              // v apunta  3*nº bytes del dato apuntado (offset), verificar

Si tenemos: float *decimal;       //suponemos que decimal apunta a  0000
decimal++;                        //decimal++ apunta a 0004

Observar las siguientes instrucciones:    int *p;   double *q;
p = &34;          // las constantes no tienen dirección
p = &(i+1);       // las expresiones no tienen dirección
&i = p;           // las direcciones no se pueden cambiar
p = q;            // error                          p = (int *)q;       // verificar
```

Compruebe:

```
void main(void)
{
int a, b, c, *p1, *p2;
void *p;
p1 = &a;                          // Paso 1. La dirección de a es asignada a p1
*p1 = 1;                          // Paso 2. p1 (a) es igual a 1. Equivale a a = 1;
p2 = &b;                          // Paso 3. La dirección de b es asignada a p2
*p2 = 2;                          // Paso 4. p2 (b) es igual a 2. Equivale a b = 2;
p1 = p2;                          // Paso 5. El valor del p1 = p2
*p1 = 0;                          // Paso 6. b = 0
p2 = &c;                          // Paso 7. La dirección de c es asignada a p2
*p2 = 3;                          // Paso 8. c = 3
printf("%d %d %d\n", a, b, c);    // Paso 9. ¿Qué se imprime?
p = &p1;                          // Paso 10. p contiene la dirección de p1
*p = p2;                          // Paso 11. p1= p2;
*p1 = 1;                          // Paso 12. c = 1
printf("%d %d %d\n", a, b, c);    // Paso 13. ¿Qué se imprime?
}
```

Punteros y arrays (opcional)
Sea el array de una dimensión:
int mat[ ] = {2, 16, -4, 29, 234, 12, 0, 3};
Cada elemento, por ser tipo int, ocupa dos bytes de memoria.
Suponemos que la dirección de memoria del primer elemento, es 1500:

&mat[0] es 1500 &mat[1] será 1502 ..... &mat[7] será 1514
El acceso mediante índices para sumar los elementos de la cuarta y sexta posición es :
x = mat[3]+mat[5]; // x = 29 + 12
Los elementos se pueden acceder por posición y por dirección, compruebe el código:

```
#include <stdio.h>  //punteros y array
#include <conio.h>  //Compruebe y analice
int mat[8]={2, 16, -4, 29, 234, 12, 0, 3}, i;  //declaración
void main()
{       printf("\n %p",&mat[0]);    // resultado: ¿? (dirección)
        printf("\n %p",mat);        // resultado: ¿? (dirección)
        i++;
        printf("\n%p",mat+i);       // resultado: ¿?
        printf("\n%d",*(mat+i));    // resultado: 16 (valor de mat[1] o valor en la dirección mat+i?
        getch();
}
```

Compruebe:
Es lo mismo &mat[0] que mat, Es lo mismo &mat[2] que mat + 2
¿Es lo mismo &mat[0] y mat ? ¿Es &mat[1] = mat++ ?

- Para pasar de un elemento al siguiente, ¿es lo mismo que a)?:

```
for(i=0; i<8; i++)  {   printf("&mat [%d] = %p", i, &mat[i]); }
```

Código a): `for(i=0; i<8; i++) { printf("mat + %d = %p", i, mat + = i); }`

Arrays de punteros
Array de punteros a cadenas de caracteres. La declaración:

```
int  *p[10];        //p es un array de 10 punteros a enteros
char *cad[10];   //cad es un array de 10 punteros a cadenas  - opcional en el curso
```

Compruebe y analice el siguiente codigo:

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
void main()
{
  clrscr(); int num,m,c,d,u;
  //            0   1   2   3   4   5   6   7   8   9
  char *letra[]={" ","y","y","y","y","y","y","y","y","y","y"};
  char *cad1[]={" ","uno","dos","tres","cuatro","cinco","seis","siete","ocho","nueve","diez","once","doce","trece",
                "catorce","quince","diesiseis","diesisiete","diesiocho","diesinueve","veinte","veintiuno",
                "veintidos","veintitres","veinticuatro","veinticinco","veintiseis","veintisiete","veintiocho",
                "veintinueve"};
  char *cad2[]={" "," "," ","treinta","cuarenta","cincuenta","sesenta","setenta","ochenta","noventa"};
  char *cad3[]={" ","cien","doscientos","trescientos","cuatrocientos","quinientos","seiscientos",
                "setecientos","ochocientos","novecientos"};
  char *cad4[]={" ","mil","dos mil","tres mil","cuatro mil","cinco mil","seis mil","siete mil","ocho mil","nueve mil"};
  printf("Ingrese un numero:\n\n");        scanf("%d",&num);       printf("resto1=%d\n",num%10000);
  m=(num%10000-num%1000)/1000;    printf("Miles=%d\n",m);  c=(num%1000-num%100)/100;
  printf("Centenas=%d\n",c);              d=(num%100-num%10)/10;  printf("Decenas=%d\n",d);
  u=(num%10-num%1)/1;             printf("Unidades=%d\n",u);
  if((num%100>0)&&(num%100<30))  { printf("%s %s %s",cad4[m],cad3[c],cad1[10*d+u]); }
    else {  printf("%s %s %s %s %s",cad4[m],cad3[c],cad2[d],letra[d],cad1[u]);  }   getch();
}

main()
{
  int *y[3][2][3],a=10;
  for (int i=0;i<3;i++)
  { for (int j=0;j<3;j++)
    { for (int k=0;k<2;k++) {  y[i][k][j]=&a;  printf("inicializando variable y[%d][%d][%d]=%d\n",i,k,j,*y[i][k][j]);  }
    }
  }
}
```

Punteros a punteros

int **p; //puntero a puntero (a un objeto int) Compruebe y analice el siguiente codigo:

```
#include "stdio.h"  //puntero a puntero
#include "conio.h"
main()
{
  int x=1;          int *p1,**p2;      p1=&x;     p2=&p1;       printf("%d %d\n",*p1,**p2);     getch();
}
```

Puntero a funcion

Un puntero a función es una variable del tipo denominado: "puntero-a-función recibiendo A argumentos y devolviendo X", donde A son los argumentos que recibe la función y X es el tipo de objeto devuelto.
Cada una de las infinitas combinaciones posibles da lugar a un tipo específico de puntero a función.
Considere detenidamente las declaraciones de los ejemplos siguientes (en todos ellos fp ó pf es un puntero a función de tipo distinto de los demás).

Observe una característica que se repite: el nombre del puntero está siempre entre paréntesis.
Ejemplos Basicos:

```
void (*pf)();  //pf es un puntero a una función, sin parámetros, que devuelve void.
void (*pf)(int);  //pfr es un puntero a función que recibe un int como parámetro y devuelve void.
int (*pf)(int, char); //pf es puntero a función, que acepta un int y un char como argumentos y devuelve un int.
int* (*pf)(int*, char*);  //pf es puntero a función, que acepta 2 punteros a int y a char como argumentos,y
devuelve un puntero a int.
```

Compruebe y analice el siguiente codigo:

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
void F2_msg(float x)  {   printf("%f\n",x); }
void F3_msg(float x)  {  printf("%f\n",x); }
void (*fp)(float);
main()
{   float pi=3.14; F2_msg(pi);      F3_msg(pi);   fp=F2_msg;  fp(pi);   fp=F3_msg;   fp(pi);    getch(); }
```

DECIMA QUINTA SEMANA:

Interfaz de usuario

Interacciones entre ser humano y máquina.
(Ver material/link entregado en plataforma virtual)

DECIMA SEXTA SEMANA: **EXAMEN FINAL**

EXAMEN SUTITUTORIO

Autor Editor:

Profesor Dr. Luis Alberto Lujan Campos

llujanc@uni.edu.pe
luislujan@neosistemas.org

2024-I