Programación Aplicada y Lab.

Jorge Rodríguez

**Apuntadores**

**Apuntador**: Es un tipo de dato que permite almacenar direcciones de memoria.

**Usos**:

* Parámetros de salida en una función.
* Listas dinámicas.
* Trabajar con variables de manera indirecta.
* Manejo de arreglos.

**Parámetros/Argumentos:**

Entrada -> por valor.

Salida -> por salida.

int main(void)

{

int numero;

numero=2;

funcion1(numero);

printf(“%d\n”,numero);

}

void funcion1(int valor)

{

valor = valor+10;

}

**Memoria** -> # Dirección y Contenido

Guarda en binario los datos.

& -> Dirección (# Memoria)

\* -> Indirección (Contenido de la dirección de memoria de la variable original, debe de declararse como pointer la variable)

numero=2

primero= &numero;

\*primero= numero+10; -> 12

\*primero= \*pnumero+2; -> 14

\*primero=\*numero+2; -> numero no está declarada como pointer -> no sirve.

**Negativo en binario:**

1.- Magnitud y signo -> el primer dígito es el que indica. 0-> + 1-> -

2.- Complemento a 1.

3.- Complemento a 2 (Hacer complemento a 1 y después sumar 1 [Volver a sumar en 1 para regresar al número binario original).

**Tipos de datos:**

**Simples** -> (enteros, de punto flotante, char), unsigned (no pone signo).

**Compuestos** -> Arreglos (Cadena) y Estructuras.

**Arreglo en una dimensión**

Arreglo es una dirección de memoria fragmentada.

int \*pnumeros;

int numeros[100]; 400 bytes (int 4 bytes \* 100)

\*numeros => numeros[0];

\*(numeros+5) => numeros[5];

pnumeros = numeros; -> se puede porque pnumeros es un pointer.

for(i=0; i<100;i++)

{

printf(“%d\n” \*pnumeros);

pnumeros++; => (aumenta en si en 4 bytes, por eso se declara en qué variable se encuentra).

}

**El Algoritmo del Programador**

**1.- Definición del problema.**

Comprender la naturaleza del problema, reconocer alcances y limitaciones.

Se mencionan las entradas, procesos, y salidas.

**2.- Planeación del problema.**

Definir el algoritmo -> Realizar el diseño del diagrama IPO / Pseudocódigo.

**3.- Codificación.**

Obtenemos un código fuente y ejecutable preliminar (Compilar y encadenador)

HW abajo de la BIOS

SW => BIOS =>SO (Administrador de Memoria, Administrador de Archivos, Sistema de Entradas y Salidas, Kernel) => APPS

gcc -> Compila y Encadena.

**4.- Pruebas y Depuración.**

Plan de Pruebas -> Detectar de Errores de Lógica o Presentación

Obtenemos un código y ejecutable definitivo (sin errores).

**5.- Documentación.**

Documentar los pasos anteriores, más el manual de usuario.

**Abstracción** => Partimos de lo general a lo particular.

**Refinación** => Comentarios para saber qué hace cada función.

**¿Qué son las estructuras?**

Las estructuras permiten a los programas agrupar tipos de datos diferentes, se conforma de una o más porciones de datos, llamadas miembros.

**¿Cómo se declaran las estructuras?**

1. El primer paso, generar una plantilla, indicar nombre y definición de miembros.
2. Para definir una estructura, debe emplearse la palabra struct.
3. Existen varias formas de declarar variables de tipo estructura.

**Ejemplo:**

**struct Def\_Alumno**

**{**

**char nombre[50];**

**char cuenta[15];**

**int cal1,cal2,cal3;**

**};**

**int main(void)**

**{**

**struct Def\_Alumno Alumno1;**

**}**

------------------------------------------------

**typedef struct def\_Alumno**

**{**

**char nombre[50];**

**char cuenta[15];**

**int cal1,cal2,cal3;**

**} TipoAlumno;**

**int main(void)**

**{**

**TipoAlumno Alumno1, Alumno2;**

**}**

¿Cómo se usan las estructuras?

Para que la estructura resulte de utilidad, es necesario tener acceso, de forma individual además de grupal, a cada miembro (o campo) de la estructura. Para lograr esto, se utiliza el operador “.”:

**No puedo leer toda una estructura, es elemento por elemento.**

**Copiar una estructura a otra si se puede -> Alumno2=Alumno1;**

**void funcion1 (TipoAlumno alumno3, TipoAlumno \*alumno4)**

**{**

**alumno3.Cal3=8;**

**(\*alumno4).Cal2=6; -> Ésta.**

**alumno4->Cal2=6; -> O ésta.**

**}**

**Serie 2 => Problema 10**

¿Qué son los archivos?

Un archivo es una colección de datos relacionados entre sí, y generalmente pensamos en información almacenada, es un flujo de datos que viene del disco.

Un archivo contiene una secuencia de Bytes, que indistintamente puede representar texto (archivos de texto), o datos binarios arbitrarios (archivos binarios).

En terminos generales

Archivos de Texto.

Archivos Binarios.

**Archivos de Texto**: Es cualquier archivo que contiene “texto plano”, y puede ser leído sin tener la necesidad de un programa o aplicación que “interprete” la información contenida en el archivo y que pueda presentarla de manera coherente, Generalmente, estos archivos están divididos en líneas.

**Archivos Binarios:** Para C, un archivo binario no es otra cosa que una secuencia de bytes.

Tipos de acceso a archivos.

En términos generales, podemos hablar de dos formas para tener acceso a la información de los archivos:

**-Acceso Secuencial.**

**-Acceso Aleatorio.**

**¿Cómo se representan los archivos en C?**

En realidad, los archivos son representados como un flujo o stream de bytes/datos.

El tipo de datos FILE.

En C, un archivo es abierto cuando el archivo físico se conecta a una stream y es cerrado cuando se rompe dicha conexión

Específicamente, cuando un archivo es abierto, C crea un objeto del tipo FILE para mantener la información necesaria para tener acceso al archivo y para tener seguimiento de su estatus.

Apuntador a FILE / Apuntador al flujo de datos

**r(leer), w(escribir), a(añadir) -> Agregar t (Texto) o b (Binario), si omitimos esto por default la máquina considerará con un t el modo de lectura.**

Error = NULL

Cerrar Archivo: fclose(Archivo);

Detectar el final del archivo: feof -> Validar si es caracter del fin del archivo, no detecta a veces correctamente el último renglón.

fprintf (Archivo, “Tengo %d años. \n”, Edad);

Abrir con w.

fscanf(Archivo, “%d %f”, &Edad, &Peso);

Abrir con rt

fputs strcpy(Cadena, “Hola a todos”);

fputs(Cadena,Archivo);

Abrir con wt

fgets char Cadena[82];

fgest(Cadena, 81, Archivo);

Cadena=”hola#\0”

Cadena[strlen(Cadena)-1]=’\0’;

Debugger en tiempo de ejecución.

Debugger Post Mortem Devuelve valores de lógica.

GDB

breakpoint -> se detiene

next -> siguiente instrucción

step -> paso a paso

print -> imprime valores de la variable

quit -> Importante cerrar

**Biblioteca String.h C**

Constantes y Tipos

|  |  |
| --- | --- |
| **Nombre** | **Descripción** |
| NULL | macro que representa la constante puntero nulo; representa un valor de puntero que no apunta a ninguna dirección válida de objeto alguno en memoria |
| size\_t | tipo entero sin signo (positivo); es el tipo devuelto por el operador sizeof |

Funciones

|  |  |
| --- | --- |
| **Nombres** | **Descripción** |
| memcpy | copia n bytes entre dos áreas de memoria que no deben solaparse  [**void \*memcpy(void \*dest, const void \*src, size\_t n)**](https://www.tutorialspoint.com/c_standard_library/c_function_memcpy.htm) |
| memmove | copia n bytes entre dos áreas de memoria; al contrario que memcpy las áreas pueden solaparse  **void \*memmove(void \*dest, const void \*src, size\_t n)** |
| memchr | busca un valor a partir de una dirección de memoria dada y devuelve un puntero a la primera ocurrencia del valor buscado o NULL si no se encuentra  **void \*memchr(const void \*str, int c, size\_t n)** |
| memcmp | compara los n primeros caracteres de dos áreas de memoria  **int memcmp(const void \*str1, const void \*str2, size\_t n)** |
| memset | sobre escribe un área de memoria con un patrón de bytes dado  **void \*memset(void \*str, int c, size\_t n)** |
| strcat | añade una cadena al final de otra  **char \*strcat(char \*dest, const char \*src)** |
| strncat | añade los n primeros caracteres de una cadena al final de otra  **char \*strncat(char \*dest, const char \*src, size\_t n)** |
| strchr | localiza un carácter en una cadena, buscando desde el principio  **char \*strchr(const char \*str, int c)** |
| strrchr | localiza un carácter en una cadena, buscando desde el final  **char \*strrchr(const char \*str, int c)** |
| strcmp | compara dos cadenas alfabéticamente ('a'!='A')  **int strcmp(const char \*str1, const char \*str2)** |
| strncmp | compara los n primeros caracteres de dos cadenas numéricamente ('a'!='A')  **int strncmp(const char \*str1, const char \*str2, size\_t n)** |
| strcoll | compara dos cadenas según la colación actual ('a'=='A')  **int strcoll(const char \*str1, const char \*str2)** |
| strcpy | copia una cadena en otra  **char \*strcpy(char \*dest, const char \*src)** |
| strncpy | copia los n primeros caracteres de una cadena en otra  **char \*strncpy(char \*dest, const char \*src, size\_t n)** |
| strerror | devuelve la cadena con el mensaje de error correspondiente al número de error dado  **char \*strerror(int errnum)** |
| strlen | devuelve la longitud de una cadena  **size\_t strlen(const char \*str)** |
| strspn | devuelve la posición del primer carácter de una cadena que no coincide con ninguno de los caracteres de otra cadena dada  **size\_t strspn(const char \*str1, const char \*str2)** |
| strcspn | devuelve la posición del primer carácter que coincide con alguno de los caracteres de otra cadena dada  **size\_t strcspn(const char \*str1, const char \*str2)** |
| strpbrk | encuentra la primera ocurrencia de alguno de los caracteres de una cadena dada en otra  **char \*strpbrk(const char \*str1, const char \*str2)** |
| strstr | busca una cadena dentro de otra  **char \*strstr(const char \*haystack, const char \*needle)** |
| strtok | parte una cadena en una secuencia de tokens  **char \*strtok(char \*str, const char \*delim)** |
| strxfrm | transforma una cadena en su forma de colación (??)  **size\_t strxfrm(char \*dest, const char \*src, size\_t n)** |

**Archivos Binarios.**

Funciones para manejo de archivos binarios

Es importante recordar que para archivo binarios en C sólo lee y escribe secuencias de bytes. Es responsabilidad del programador, saber cómo deben ser interpretadas dichas secuencias.

En las siguientes láminas veremos algunas funciones para manejo de archivos binarios:

* fwrite
* fread

Escribir una secuencia de bytes en un archivo -fwrite

Sintaxis

si<e t\_ fwrite (void \*buffer, size\_t size, size\_t number, FILE \*file);

Uso: Escribe una secuencia de bytes en el archivo \*buffer es un pointer a los datos que van a ser descritos, size es el tamaño de bytes del elemento que voy a escribir y number es el número de elementos de tamaño size que deseo escribir. (fwrite escribe size\*number bytes).

Valor de la función: Devuelve el valor de number si la escritura fue correcta, un número inferior en caso contrario.

Ejemplo:

int Numero, Lista[10];

fwrite(&Numero, sizeof(int), 1,Archivo);

/\*Escribe el valor de la variable Numero en el archivo\*/

fwrite(Lista,sizeof(int),10,Archivo);

/\*Escribe todo los valores del arreglo\*/

Leer una Secuencia de bytes de un archivo -fread.

**Valor de la función:** Devuelve el valor de number si la lectura fue correcta, un número inferior en caso contrario.

**Ejemplo:**

int Numero, Lista[10];

fread(&Numero, sizeof(int),1,Archivo);

fread(Lista,sizeof(int),10,Archivo);

Movernos en el archivo:

fseek (archivo, 0L=>número de bytes a desplazarse, [SEEK\_SET/SEEK\_END/SEEK\_CUR]=>posición desde la cual desplazarse)

0L= puede ser variable (debe ser long int) o número.

SEEK\_SET => moverse desde 0 bytes.

SEEK\_END => irse hasta el fin del archivo.

SEEK\_CUR => moverse desde la posición de donde estoy.

ftell(Archivo) => Cantidad de bytes en el que estoy posicionado del archivo (long int).

**Argumentos de Programa**

int main (int argc, char \*argv[])

argc= # de elementos que hay en la línea de comandos (incluye el nombre del programa).

argv-> Arreglo de cadenas, donde están todos los parámetros introducidos.

**Asignación Dinámica de Memoria.**

Dimensionar el arreglo de manera dinámica.

malloc => Asigna Espacio de Memoria

malloc(número de bytes) => retorna la dirección de memoria donde me reservó el bloque, pero necesita declarar de qué forma quieres recibir los datos (int \*), (float \*), (char \*), (TipoElemento \*).

**Ejemplo.**

int Numtemps,\*Listatemps;

printf(“Cuantas temperaturas quieres?\n”);

scanf(“%d”,&Numtemps);

ListaTemps=(int \*)malloc(sizeof(int)\*numtemps);

for( int i=0; i<Numtemps; i++)

{

printf(“Ingrese la temperatura [%d]: \n”,i+1);

scanf(“%d”, &ListaTemps[i]);

}

free(ListaTemps);

**Listas Dinámicas.**

Estructuras Dinámicas => Listas-> pilas(Primera que se puso es la última en ser retirada) [LIFO] || colas (Impresora) [FIFO] || listas [Encadenación sencilla o doble] || Árboles

FIFO && LIFO => PUSH, POP

**Listas con encadenación sencilla.**

Los asigna en un solo bloque y nos regresa la dirección del primer elemento.

El último elemento debe estar siempre asignado con NULL para determinar el término de la lista dinámica.

Y tener una variable de que tenga la dirección de memoria al primer elemento.

Pasos a seguir para agregar un nuevo elemento al **Inicio**:

1. Crear el nuevo elemento [new (temp)].
2. Asignarle valor del nuevo elemento. temp->dato=valor;
3. Encadenar a la lista. temp.sig=Inicio;
4. Reservar el inicio de la lista. Inicio=temp;

Pasos a seguir para agregar un nuevo elemento al **Fin**:

1. new (temp);
2. temp->dato=valor;
3. temp->sig=NULL;
4. new(temp2);
5. if(\*Inicio<>NULL)
6. temp2=Inicio;
7. mientras(temp2->sig<>NULL)
8. temp2=temp2->sig;
9. temp2->sig=temp;
10. sino
11. \*Inicio=temp;

Pasos para **Imprimir** todos los elementos de la lista:

1. new(temp);
2. temp=Inicio;
3. mientras(temp<>NULL)
4. {
5. Imprimir Elementos;
6. temp=temp->sig;
7. }

Pasos para **Liberar** los elementos de la lista: (Salida Inicio)

1. new(temp);
2. mientras(Inicio<>NULL)
3. {
4. temp=Inicio;
5. Inicio=Inicio->sig;
6. free(temp);
7. }

Para conservar los valores de un apuntador (se necesita un doble apuntador).

Pŕactica 1

Encabezado es: p5 largo ancho 255 \_ ->valores de los pixeles

Buscar el valor mas pequeño y el mas grande -> ecualización

Menor valor en un 0. (min,0)

Mayor -> convertir a 255. (max,255)

Intermedios -> ecuación de la recta conociendo dos puntos

Y es el nuevo valor

mx+b

Reducir -> quitar la mitad de las columnas y quitamos la mitad de los renglones

Aumentar -> lo opuesto.

1px en 4px

1 Semana de Diseño

2 Semana Codificar

Formato EEE

Anexo-> Diagrama IPO

**Módulos**

extern para compartir funciones / valores de una variable de un archivo a otro.

Archivo: makefile

comandos: make ó make (función)

Archivos dentro del directorio.

**Recursión**

Una función dentro de una función, no es un ciclo.

Sólo usar en procesos recursivos.

int main (void)

{

int a,b;

a=0;

b=10;

* Marca para regresar. funcion1(a, b);

.

**.**

**}**

**funcion1 (int x, int y){**

**int c;**

**.**

**.**

* **Marca para regresar funcion1(x,y);**

**}**

int factorial (int numero)

{

if (numero>1)

return numero \* factorial (numero-1);

else

return 1;

}

**Listas Circulares**

* **El último apunta al primero**
* **La encadenación doble el último apunta a Inicio e Inicio Apunta a último.**

**Árboles binarios.**

Estructura dinámica que nos permite clasificar datos, la información conforme estemos trabajando la mantendremos en orden.

Rama izquierda y rama derecha, máximo de dos ramas.

Nodo raíz, subárbol, derecho e izquierdo, estructura recursiva

Partimos del nivel 0, hasta donde termine el caso.

Los últimos nodos, son llamados nodos hojas.

Árbol completo cuando tenga todas las ramas completas.

Nivel del balanceo del árbol

B= longitudnivelizquierda- longitudnivelderecha.

Árbol Balanceado cuando B = -1, 0, 1 No más, no menos.

Raíz = NULL

Si el número es menor o igual al nodo raíz inserto a la izquierda.

Imprimir de manera recursiva.

Orden para recorrerlo= todo elemento a la izquierda, después todo elemento a la derecha.

Métodos para Insertar

1.- No recursivo:

return; ->Cuando inserta, termina con el return;

Recorrer en orden:

Izquierda-> Imprimo -> Derecha.

Recorrer en Preorden:

Imprimir -> Izquierda -> Derecha

Liberar Memoria en Post Orden:

Izquierda -> Derecha -> Borrar

Entre menos probabilidad de mensaje mayor información obtengo.

Entre mayor probabilidad de mensaje menos información obtengo.

Código Comparar de Abajo hacia Arriba.

Método de ordenamiento.

Algoritmos directos e indirectos.

Directos -> Más simples, pero menos eficientes.

* 4 ) Directa -> una sola función con arreglos, trabajar con archivos binarios, en sí en estructuras estáticas. Menos eficiente comparar todos los números
* 2) Burbuja -> Menos ciclos n a la 2
* 3) Selección->
* 1) Inserción.

Indirectos -> Menos simples, pero más eficientes.

* 2) Quicksort
* 1) Heapsort

Programación Orientada a Eventos.

Cada elemento tiene características y comportamiento.

**Esquema**

Vista <-> Controlador <-> Modelo

Vista -> Código para generar la parte gráfica.

Modelo -> Código para que funcione

Controlador -> Conexión de Vista y Modelo. / Punto de interacción entre los dos estados.

Librería GTK

https://developer.gnome.org/gtk2/stable/

Librería GDB

Compilar GTK

./compila nombre del archivo sin extensión.

Ejemplo:

./compila helloworld

gtk main(); -> Entra a un loop infinito

Herencia.

Una ventana va a heredar cosas del ambiente gráfico