Manual de Prácticas del Laboratorio de Programación de Computadoras.

M.I. Moisés García Villanueva.

23 de septiembre de 2010

Introducción

Este manual pretende apoyar la materia de Laboratorio de Programación de Computadoras con ejercicios prácticos sencillos, que le permitan al alumno incrementar sus habilidades en el conocimiento del lenguaje de programación C.

El manual consiste de 14 prácticas, cada una de ellas indica una serie de actividades que le proporcionará al estudiante las habilidades de la temática que se pretende cubrir.

El programa de la materia de Laboratorio de Computadoras está compuesto de 14 prácticas y 2 evaluaciones.

Práctica 1

Introducción al lenguaje C

1.1. Requerimientos de la práctica.

Debemos entender cual es el procedimiento para lograr que una computadora efectue el trabajo que deseamos mediante instrucciones del **lenguaje C**. La figura 1.1 nos muestra los diferentes pasos que debemos seguir en el proceso de la programación.

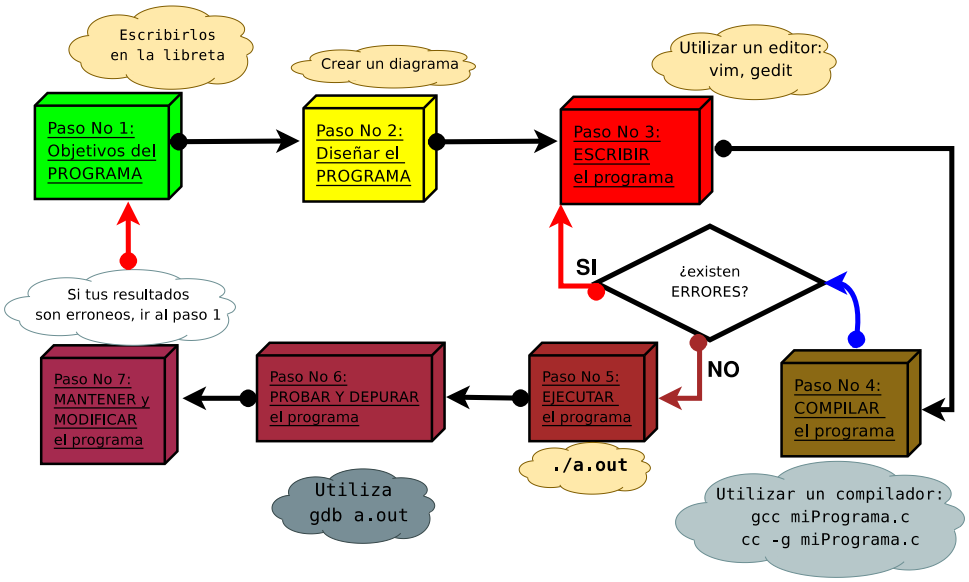

Figura 1.1: Los siete pasos en el proceso de programación.

Ahora practiquemos cada uno estos pasos que se requieren para iniciarse como programadores de un lenguaje de programación, en nuestro caso lenguaje C.

1.2. Actividades de la práctica.

1.2.1. Paso No. 1: Objetivos del programa.

En este paso, vamos a plantear diferentes problemas básicos y escribiremos los objetivos que se prentenden del programa.

- Problema 1. Deseamos crear un programa que muestre en la pantalla un letrero que diga "`Mi primer Programa en C`".
  Los objetivos del programa los vamos a obtener a partir de la definición o planteamiento del problema que se nos plantea. Para ello es bueno que te vayas acostumbrando a leer por partes el enunciado que se

te da y tratar de entender cada una de las palabras. Los objetivos son aquello que te piden realizar en tu programa.

El objetivo principal del problema 1 es entonces: **Mostrar o imprimir en pantalla un letrero**.

- Problema 2. Dada una secuencia de números, muestre todos aquellos números que son pares.
  Objetivos:
  - Generar una secuencia de números. $\{ 1, 2, \ldots n\}$
  - Determinar los que son pares.
  - Imprimir en pantalla todos los números pares.

1.2.2. Paso No. 2: Diseño del Programa.

En el diseño del programa se pueden utilizar:

1. **Diagramas de flujo**. Los diagramas de flujo son figuras que representan acciones. El diseño del programa con estas figuras es ir armando la secuencia de pasos o acciones que me permitan realizar los objetivos del programa.
2. **Seudocódigo**. Esté método de diseño lo que hace es indicar las acciones a realizar mediante palabras del idioma que nosotros manejamos. Se debe decir con palabras cada uno de los pasos que debe realizar nuestro programa para lograr los objetivos del programa.

Algunas figuras de los diagramas de flujo con sus respectivas acciones se muestran en la figura 1.2.

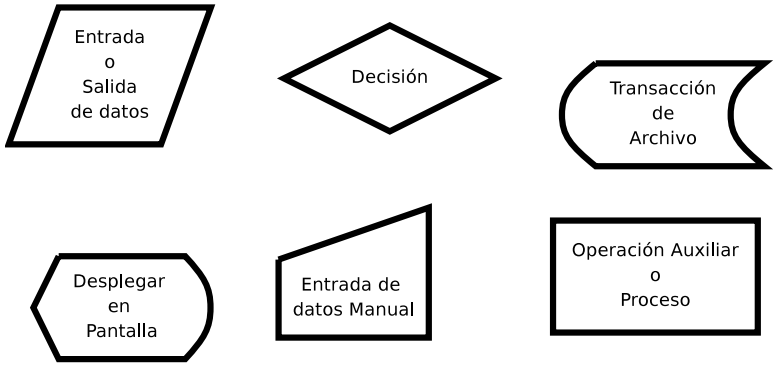

Figura 1.2: Figuras que representan acciones en los diagramas de flujo.

- El diagrama de flujo para un problema en donde el objetivo es imprimir en pantalla un letrero se muestra en la figura 1.3. En el diagrama observamos los indicadores de inicio y fin del programa, dentro de los cuales se encuentran las instrucciones que me permitan realizar los objetivos.
- Se puede tener más de un diagrama de flujo para la solución de un problema, en la figura 1.4 se muestran dos diagramas de flujo que nos permiten resolver el problema 2 planteado en la sección 1.2.1. En el bloque de operaciones de la figura 1.4 (b) nos encontramos con el operador %, denominado como `modulo`, el cual nos proporciona el valor del residuo que se obtiene al dividir el valor de *secuencia* entre 2. Todo numero dividido entre 2 nos indica que es par.

1.2.3. Paso No. 3: Escribir el Programa.

El siguiente paso en el proceso de programación se refiere a la escritura del programa, es decir, los bloques que se incluyeron en el diagrama de flujo se traducen a instrucciones del lenguaje C. En el caso del pseudocódigo se refiere a traducir las palabras del idioma a palabras o instrucciones del lenguaje C.

Cualquier editor de textos que nos proporcione un archivo de caracteres ascii lo podemos utilizar para la etapa de edición o escritura de programas. En las siguientes actividades vamos a realizar la escritura de un programa a partir de como ejecutar el editor de textos.

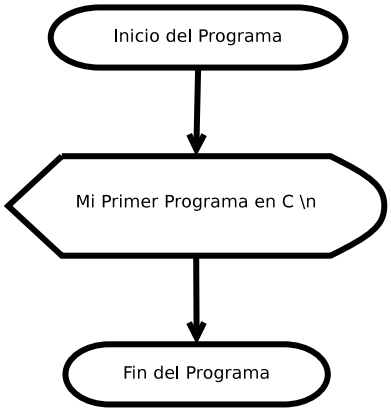

Figura 1.3: Diagrama de flujo para imprimir un letrero en pantalla.

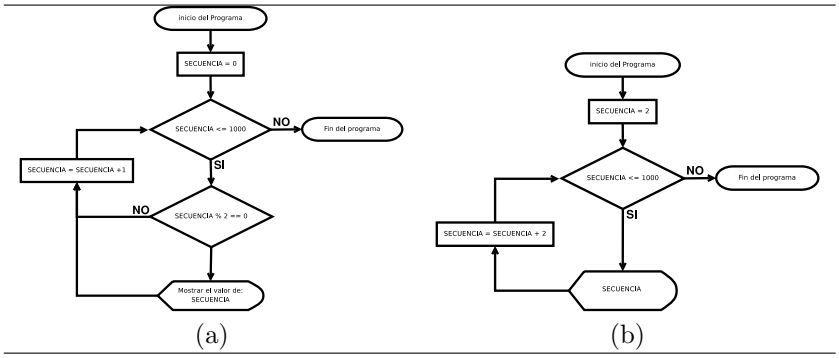

Figura 1.4: Dos posibilidades de diagrama de flujo para imprimir los números pares de una secuencia de números.

Actividades.

1. Presiona las teclas al mismo tiempo `Alt F2` para ejecutar una aplicación del sistema, en el campo de texto escribe `xterm` y presiona la tecla `Entrar` o `Enter`.
2. Ahora escribe el nombre del editor que quieres utiliar para editar tus programas:
   a) La primer opción puede ser: `vim`
   b) La segunda opción puede ser: `gedit`
3. Una vez dentro del editor escribimos las siguientes lineas de código del lenguaje C y guardamos el archivo con el nombre `programa1.c`, este programa lo único que hace es imprimir en la pantalla un letrero.

```
#include <stdio.h>

void main(void){

    printf("Mi primer programa en el lenguaje C\n");

}
```

4. Para el diagrama de flujo de la figura 1.4 (b) creamos un nuevo archivo que se llame `problema2.c` y agregamos las siguientes líneas de código:

```
#include <stdio.h>
```

```
void main(void){
        int i;

        i=2;

        while( i <= 1000 ){
             printf("%d ",i);
             i = i + 2;
        }
}
```

Indica en ambos casos la equivalencia de instrucciones o lineas de código con lo bloques del diagrama de flujo.

1.2.4. Paso No. 4: Compilar el Programa

El siguiente paso después de la edición es la compilación, para ello abrimos una nueva terminal para ejecutar comandos; presiona las teclas `Alt F2` al mismo tiempo y escribe en el campo de texto `gnome-terminal`

Sobre la nueva terminal verificamos que el archivo `programa1.c` se encuentra en el directorio actual, escribe el comando `ls` y presiona `enter`; busca el nombre del archivo `programa1.c` dentro de la lista de archivos que se desplego en pantalla. Si encontrastes el archivo en la lista continua con el siguiente paso de la práctica, en caso contrario utiliza los comandos `find` o `locate` para localizar tu archivo. Es importante ubicarse dentro del directorio en donde se encuentra nuestro archivo a compilar, en este ejercicio el archivo a compilar se llama `programa1.c`

La compilación de un programa se realiza mediante el comando `gcc` o `cc`. A continuación se muestran diferentes opciones para compilar el archivo `programa1.c`, puedes experimentar escribiendolas en la terminal de comandos.

- `gcc programa1.c`
  Con esta opción la compilación del programa me genera un archivo de salida que se llama `a.out`, el cual esta listo para ejecutarse en la computadora.

- `gcc -g programa1.c`
  Con la opción de `-g` podemos DEPURAR la ejecución de nuestro programa, es decir, verificar paso a paso la ejecución de cada instrucción que escribimos en el programa.

- `cc programa1.c -o programa.ejecutable`
  Para decidir que nombre de archivo de salida deseamos nosotros para nuestro programa utilizamos la opción de `-o` al momento de la compilación, con ello estamos realizando una asignación de nombre al archivo de salida o archivo de ejecución, en este caso el archivo de salida `a.out` se substituye por el archivo `programa.ejecutable`

- `cc -g -o archivo.ejecutable programa1.c`
  Podemos incluir varias opciones en la compilación, en este caso el nombre de archivo de salida o ejecución lo queremos llamar `archivo.ejecutable` y además podemos DEPURAR o verificar la ejecución de este programa paso a paso.

1.2.5. Paso No. 5: Ejeutar el Programa.

Ahora procedemos a EJECUTAR nuestro programa con la finalidad de verificar si realmente hace lo que nosotros codificamos en la parte de EDICIÓN. La forma de ejecutar el programa es anteponiendo la secuencia de caracteres `./` al nombre de archivo ejecutable que se genero en la etapa de COMPILACIÓN. Para el caso que hemos venido desarrollando según la opción de compilación que utilizamos escribimos lo siguiente en la terminal de comandos:

- `./a.out`
- `./programa.ejecutable`
- `./archivo.ejecutable`

1.2.6. Paso No. 6: Probar y Depurar el Programa.

En esta etapa del proceso de programación, es posible verificar la ejecución del programa línea por línea o instrucción por instrucción. El programa que nos permite realizar este paso es `gdb` que proviene de GNU Debugger es el depurador estándar para el sistema operativo de licencia libre. GDB ofrece la posibilidad de trazar y modificar la ejecución de un programa. El usuario puede controlar y alterar los valores de las variables internas del programa. Una propiedad que al momento de iniciarse en el apréndizaje del lenguaje no nos favorece es que GDB no contiene su propia interfaz gráfica de usuario y por defecto se controla mediante un interfaz de línea de comandos.

Para poder examinar el programa paso a paso, es necesario compilarlo con la opción `-g` y de esta forma podemos ejecutar en la terminal de comandos del sistema operativo lo siguiente:

```
usuario@lc40\$ gdb a.out
```

En el caso de haber utilizado la opción de `-g -o programa.ejecutable` la ejecución para depurar el programa será:

```
\item usuario@lc40\$ gdb programa.ejecutable
```

Algunos de los comandos más comunes en GDB y que es necesario memorizarlos se listan a continuación:

- `break numero_linea`. La primer acción que debemos indicar al momento de depurar un programa es indicar el número de línea en en que deseamos detener el programa, esto lo logramos con el comando `break` e indicando el numero de línea en donde se desea detener el programa.
  Otra opción para indicar el punto de paro es indicar el nombre de la función, por lo tanto la sintaxis para realizarlo será: `break nombre_función`.
- `run`. Este es el comando para iniciar la ejecución del programa, después de indicar este comando dentro de la línea de comandos del `gdb` la ejecución del programa se detendrá en la posición indicada en el comando `break`.
- `display nombre_variable`. Esta instrucción nos permite la visualización del valor que actualmente tiene la variable que se indica, `nombre_variable` se refiere a cualquiera de las variables que se definieron en el programa.
- `next`. Esta instrucción nos permitirá ejecutar la siguiente instrucción del programa que estamos depurando.
- `continue`. Con esta instrucción indicamos que la jecución del programa se continue hasta encontrar un nuevo punto de paro o hasta encontrar el final del programa.
- `quit`. ¿Como salir del programa `gdb`?, para ello escribimos el comando `quit` y salimos del proceso de depuración de nuestro programa.

Actividades

1. Compila el programa que resuelve el problema 2 de la sección 1.2.1 de la siguiente forma:

   ```
   gcc -g problema2.c
   ```

2. Ahora ejecutamos el `gdb` para depurar nuestro programa de la siguiente forma:

   ```
   gdb a.out
   ```

3. La primer instrucción es indicar el punto de paro, escribimos:

   ```
   break main
   ```

4. Indicamos que se inicie la ejecución del programa:

   ```
   run
   ```

   En este punto observamos que la ejecución del programa se detuvo en la instrucción `i=0`.

5. Ahora veamos el valor que tiene la variable `i` con el comando `display`, para ello escribimos en línea de comandos lo siguiente:

   ```
   display i
   ```

6. Presionamos varias veces la instrucción `next` o `n` y observamos como el valor de la variable `i` va cambiando.

   ```
   next
   n
   n
   next
   ```

7. Finalmente indicamos a `gdb` que continue con la ejecución del programa hasta terminar.

   ```
   continue
   ```

1.2.7. Paso No. 7: Mantener y Modificar el Programa.

En este paso debemos verificar si los resultados o cálculos realizados por el programa son los esperados, en caso de existir error debemos verificar a partir del paso No. 1 todo el trabajo realizado.

Práctica 2

Tipos de Datos, operadores, enumeraciones, expresiones y sentencias.

Palabras del lenguaje por conocer en el tema:

- int, float, double, char, unsigned, long.

La estructura general para definir los diferentes tipos de datos según corresponda son:

```
[signed|unsigned] char <identificador>[,<identificador2>[,<identificador3>]...];

[signed|unsigned] [short|long] int <identificador>[,<identificador2>[,<identificador3>]...];
[signed|unsigned] long [int] <identificador>[,<identificador2>[,<identificador3>]...];
[signed|unsigned] short [int] <identificador>[,<identificador2>[,<identificador3>]...];

float <identificador>[,<identificador2>[,<identificador3>]...];

[long] double <identificador>[,<identificador2>[,<identificador3>]...];
```

Actividades

1. En un editor de textos, escribir la estructura básica de un programa en C.
2. Dentro de la etapa de instrucciones, escribir la definición de variables enteras, en las cuales se almacenaran los valores para las siguientes expresiones:

$$a = b + c$$

$$a = b - c * 1{,}13452d = \frac{a}{b}$$

$$a = \frac{b}{c}d = (1{,}00023 * a) + \frac{c}{a}$$

   Para los valores $b = 3$ y $c = 7$ indica el porcentaje de error en los cálculos en cada una de las expresiones.
3. Ahora utiliza el tipo de dato $float$ en las variables definidas en la actividad anterior e indica el porcentaje de error nuevamente en las expresiones dadas.
4. Indica el tipo de dato que corresponde al utilizar a cada uno de los siguientes ejemplos de información o acciones que se desea utilizar en un programa de lenguaje C:

- Número de habitantes: _____
- Valores booleanos (cero o uno): _____
- Velocidad en metros por segundo: _____
- Operaciones con π: _____
- Número de autos: _____
- Operaciones con funciones trigonométricas: _____
- número de letras: _____
- Comparar letras: _____

5. Con el `gdb` (paso de depuración de un programa), verifica los valores que se van asignando a cada una de las variables.

6. Crear un programa en donde se utilize una expresión matemática con tipos de datos flotante y entero. Los resultados de la expresión matemática al utilizar los diferentes tipos de datos se deben imprimir en la pantalla en columnas diferentes, por ejemplo para la siguiente expresión matemática:

$$R = (Va - Vb)/I$$

va	vb	i	Va	Vb	I	R(int)	R(float)
10	2	2	10.000	2.000	2.000	4	4.000000
10	2	2	10.000	2.000	2.000	4	4.000000
9	2	2	9.800	2.200	2.200	3	3.454545
8	2	2	9.400	2.600	2.600	3	2.615385
7	2	2	8.800	3.200	3.200	2	1.750000
6	2	2	8.000	4.000	4.000	2	1.000000
5	3	3	7.000	5.000	5.000	0	0.400000
3	4	4	5.800	6.200	6.200	0	-0.064516
1	5	5	4.400	7.600	7.600	0	-0.421053
0	6	6	2.800	9.200	9.200	-1	-0.695652
-1	7	7	1.000	11.000	11.000	-1	-0.909091

Determina el porque incremento o decremento en cada una de las variables `va`, `vb`, `i`, `Va`,`Vb` e `I` que como se observa en la tabla las primeras tres son del tipo entero y las otras son del tipo flotante. El incremento o decremento que se aplico fue el mismo, ¿porque se obtienen esos resultados?.

Práctica 3

Entrada de datos y salida por consola.

Palabras del lenguaje por conocer en el tema:

- `getchar();` Función que nos permite leer un caracter o dato desde el teclado o el flujo de entrada.
- `putchar();` Función que nos permite escribir en la pantalla un caracter o un dato. Genera un flujo de datos de salida.
- `scanf(‘‘formato de entrada’’,variables)`
- `printf(‘‘formato de salida’’,variables)`

Las funciones que realizan la entrada y salida de datos **SIN FORMATO** son:

- `getchar()`: Lee un carácter del teclado. Espera hasta que se pulsa una tecla y entonces devuelve su valor.
- `putchar()`: Imprime un carácter en la pantalla en la posición actual del cursor.
- `gets()`: Lee una cadena de caracteres introducida por el teclado y la sitúa en una dirección apuntada por su argumento de tipo puntero a carácter.
- `puts()`: Escribe su argumento de tipo cadena en la pantalla seguida de un carácter de salto de línea.

Las funciones principales que realizan la entrada y salida **CON FORMATO**, es decir, se pueden leer y escribir en distintas formas controladas, son:

- `printf()`: Escribe datos en la consola con el fo rmato especificado.
- `scanf()`: Función de entrada por consola con el formato especificado.

El siguiente ejemplo ilustra el uso de funciones sin formato al leer un caracter de entrada e imprimirlo en pantalla.

```
#include<stdio.h>

main(){
     char letra;

     letra=getchar();

     putchar(letra);

}
```

Ahora veamos un ejemplo del uso de funciones de entrada y salida con formato. El formato de entrada es el siguiente:

```
a0920312g : Juan González : 123.80
```

y el formato de salida deseado de los datos es:

```
Nombre:         Juan González
Matrícula:      a0920312g
Ahorro:         $123.80000
```

El programa que me lee los datos con el formato dado y genera la salida correspondiente es el siguiente:

```
#include <stdio.h>

main(){

    char matricula[10];
    char nombre[10];
    char apellido[10];
    float ahorro;
    char separador;

    scanf("%s :",&matricula);
    scanf("%s %s :",nombre,apellido);
    scanf("%f",&ahorro);

    printf("Nombre:\t\t%s %s\nMatrícula:\t%s\nAhorro:\t\t$%f\n",nombre,apellido,matricula,ahorro);
}
```

Actividades

1. Escribir las funciones prototipo de: `putchar(), getchar(),gets(), puts()`.
2. Hacer un programa que lea del teclado 5 caracteres y los imprima en pantalla al concluir la lectura de los 5 caracteres.
3. Hacer un programa que lea caracteres hasta que se presione el caracter `@`.
4. Hacer un programa que lea desde el teclado números complejos de la forma: `(3,2j)`, es decir: `(parte_real,parte_imaginaria j)`.
5. Hacer un programa que lea líneas completas de texto y las duplique, hacer uso de las funciones `gets()` y `puts()`.

Práctica 4

Instrucciones condicionales

Palabras del lenguaje por conocer en el tema:

- if . . . else
- if . . . else if . . . else
- switch . . . case . . . default

Constantemente se deben tomar decisiones al momento de programar, ello determina acotar la solución del problema por resolver. La figura 4.1 nos muestra el diagrama de flujo para una condición.

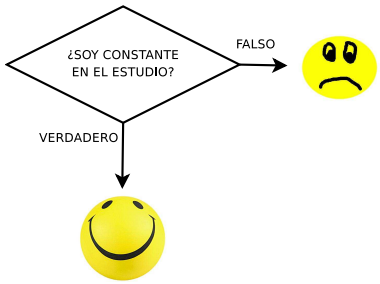

Figura 4.1: Diagrama representativo de condicionales. La respuesta a la pregunta realizada toma los valores de verdadero o falso.

La figura 4.2 nos muestra los diferentes operadores en las instrucciones condicionales y ejemplificando su uso con algunas variables.

Para tomar múltiples decisiones sobre un mismo tipo de dato en la entrada tenemos la opción del selector múltiple `switch...case`, la figura 4.3 nos ilustra en forma gráfica la acción de un selector de opciones, en donde se indica que dependiendo del objeto que logra introducirse o aceptarse en el selector se realizan las acciones correspondientes.

Actividades

1. Hacer un programa que lea 3 números enteros desde teclado e indique cual es el menor y el mayor de los tres.
2. Hacer un programa que lea dos palabras e indique cual de las dos contiene el mayor número de letras.
3. Hacer un programa que lea desde teclado una fecha e indique la estación del año a la que corresponde la fecha.
4. Realizar un menú con las diferentes opciones que se requieren en una agenda, como son: dar de alta un contacto, eliminar un contacto, buscar un contacto, modificar un contacto y salir de la agenda.

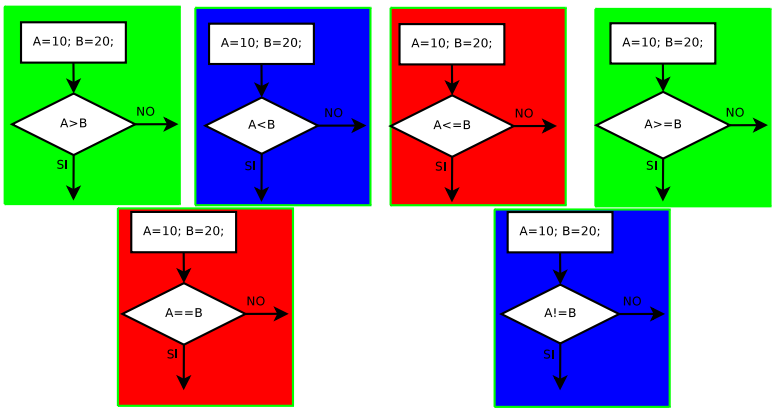

Figura 4.2: Diferentes operadores en las condicionales básicas.

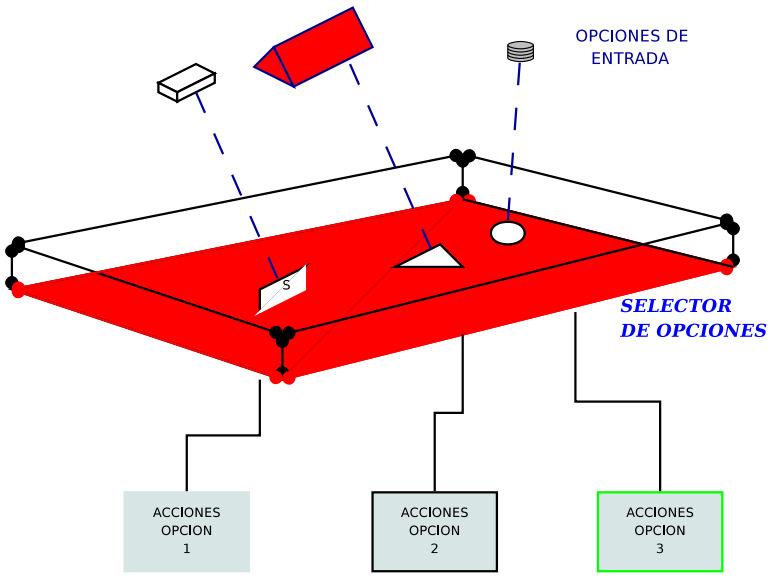

Figura 4.3: Representación gráfica de un `switch...case`.

Práctica 5

Instrucciones de Repetición

Palabras del lenguaje por conocer en el tema:

- `for`(inicialización; condición; incremento)
- inicialización; `while`(condición) incremento
- inicialización; `do`; instrucciones;incremento; `while`(condición)

En la programación el uso de ciclos es de suma importancia, por ello es necesario conocer perfectamente las estructuras básicas de los diferentes tipos de ciclos en el lenguaje.

Estructura básica del ciclo `for`

```
for ( inicialización; condición; incremento; )
{
     instrucciones_del_lenguaje;
}
```

Estructura básica del ciclo `while`

```
inicialización;
while (  condición )
{
     instrucciones_del_lenguaje;
     incremento;
}
```

Estructura básica del ciclo `do-while`

```
inicialización;
do
{
     instrucciones_del_lenguaje;
     incremento;
}while (  condición );
```

El diagrama de flujo de un ciclo lo podemos representar mediante la figura 5.1, la cual nos ilustra un ejemplo de un ciclo, mencionando que el incremento puede tomar alguna de las tres opciones mostradas. Es importante resaltar que un ciclo cuenta con cuatro componente: **INICIALIZACION, CONDICION, INSTRUCCIONES** e **INCREMENTO** que son las partes que el programador debe modificar para implementar el ciclo que nos resolverá algún trabajo.

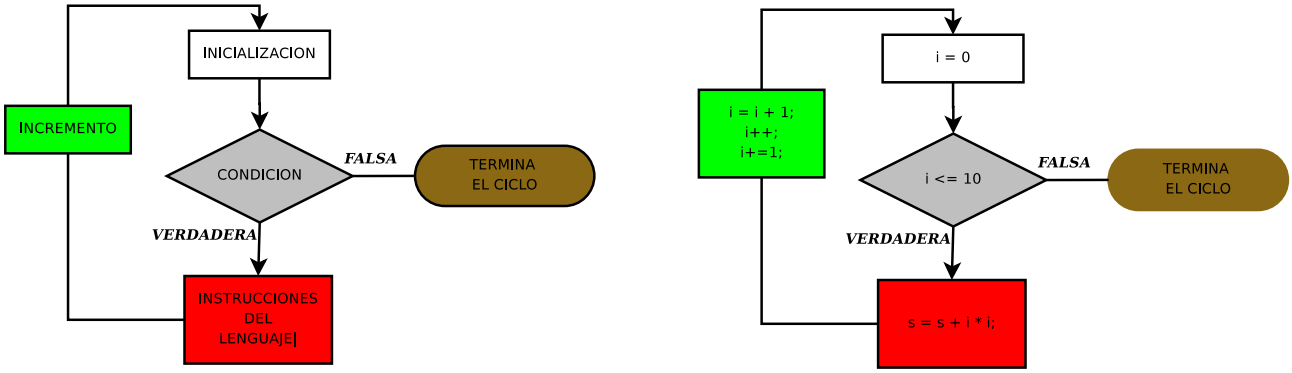

Figura 5.1: Representación gráfica de un ciclo, en la figura se presenta la estructura general de un ciclo (izquierda) y un ejemplo ilustrando todos los componentes (derecha).

Actividades

1. Escribe en la figura 5.2 las partes del ciclo de acuerdo a las siguientes instrucciones:

   a) `variable1 = 3.25;`

   b) `variable1 == 10.75;`

   c) `variable1 += 0.25;`

   d) `variable2 += variable1;`

   e) `variable2 = variable2 + 0.75;`

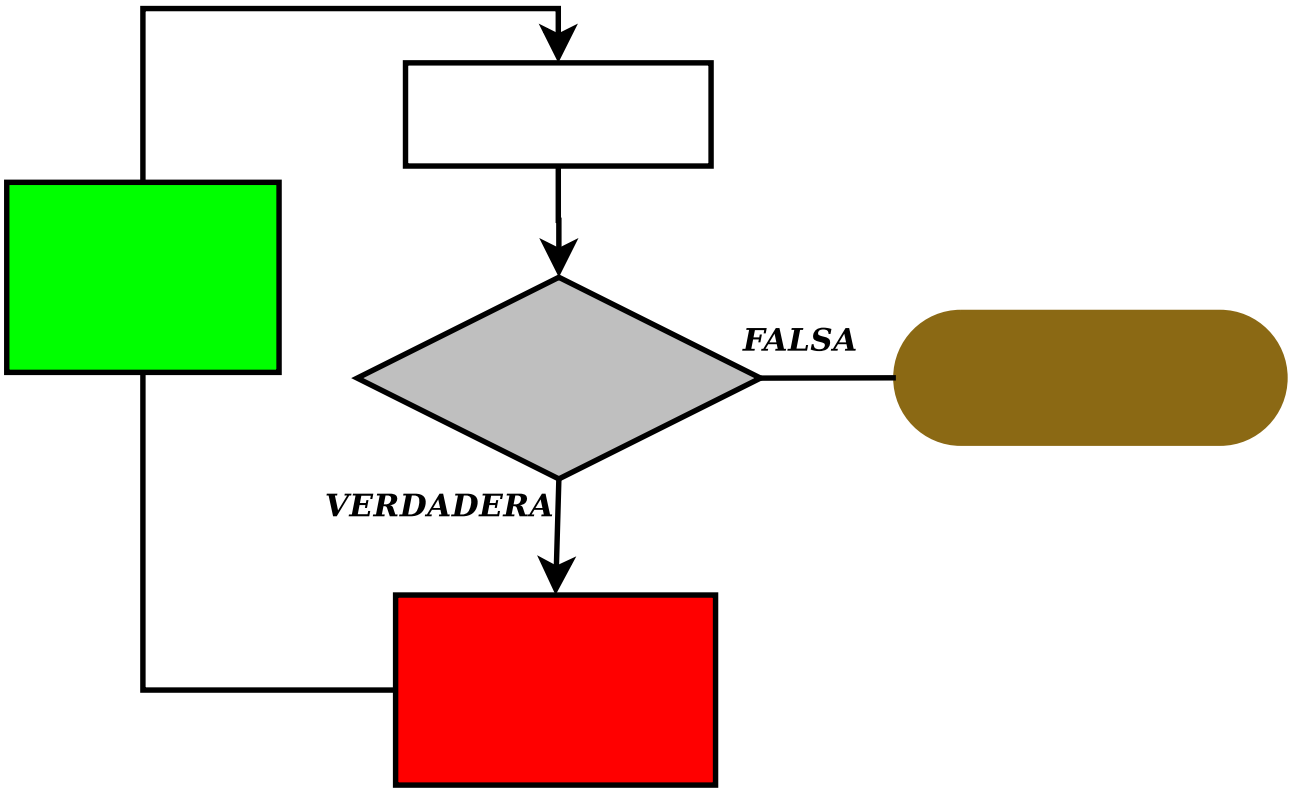

Figura 5.2: Escribir las instrucciones de la actividad 1 según corresponda en el diagrama de flujo que representa un ciclo.

2. Obten el diagrama de flujo del ciclo que se encuentra en el siguiente programa e indica que hace el programa.

```
#include <stdio.h>
int INCREMENTO=12;

main(){
      int i=120;

      while (i > 0){
            printf("%d\n",i);
            i -= INCREMENTO;
      }
}
```

3. Obten el diagrama de flujo del ciclo que se encuentra en el siguiente programa e indica que hace el programa.

```
#include <stdio.h>

main(){
      int i;

      for (i=0;i == 1110; i+=1){
            if ( (i % 2) == 0 ){
                  printf("%d\n",i);
            }
      }
}
```

4. Hacer un programa que indique cuantos caracateres `b` se encuentra en el flujo de texto de entrada, la lectura de caracteres termina hasta encontrar el caracter `EOF`.

5. Hacer el diagrama de flujo de un programa que lee caracter por caracter hasta encontrar el indicativo `EOF` y en el proceso cuenta las vocales minúsculas que existen en el texto.

Práctica 6

Programación estructurada usando funciones

Primero indicaremos la estructura básica de una función en el lenguaje C.

```
tipo_dato_regresa nombre_de_la_función ( lista_de_argumentos_o_parametros_de_entrada )
{
intrucciones_del_lenguaje;

return variable;  // si el tipo de dato que regresa es void esta parte se suprime.
}
```

- `tipo_dato_regresa` se refiere al tipo de dato o valor que regresa la función, el cual puede ser cualquir tipo de los descritos en la práctica 2 o `void`, este último se refiere a que la función no regresa NADA.
- `nombre_de_la_función` puede ser cualquier nombre que no sea palabra reservada del lenguaje o nombre de función existente en las bibliotecas. Se recomienda que el nombre de la función se relacione a la acción o problemática que resuelve.
- `lista_de_argumentos_o_parametros_de_entrada` se refiere a la definición de varialbes de entrada, cuando una función es llamada o invocada por el programa debe recibir el número de argumentos con que fue declarada y además del mismo tipo de dato. En caso de que la función no reciba argumentos se debe utilizar la palabra reservada `void`.

Para poder utilizar una función en nuestro programa es necesario seguir las siguientes reglas:

1. **Declaración de la función.** La declaración de una función se conoce también como prototipo de la función. En el prototipo de una función se tienen que especificar los parámetros de la función, así como el tipo de dato que regresa. Los prototipos de las funciones que se utilizan en un programa se incluyen generalmente en la cabecera del programa y presentan la siguiente sintaxis:

   ```
   tipo_dato_regresa nombre_de_la_función ( lista_de_argumentos_o_parametros_de_entrada );
   ```

2. Definición de la función. Después de declarar una función, el siguiente paso es implementarla. Generalmente, este paso se conoce como **definición**. Es precisamente en la **definición** de una función donde se especifican las instrucciones que forman parte de la misma y que se utilizan para llevar a cabo la tarea específica de la función. La **definición** de una función consta de dos partes, el **encabezado** y el **cuerpo** de la función. En el **encabezado** de la función, al igual que en el prototipo de la misma, se tienen que especificar los parámetros de la función, si los utiliza y el tipo de datos que regresa, mientras que el **cuerpo** se compone de las instrucciones necesarias para realizar la tarea para la cual se crea la función, esta delimitado por los carcateres { y }. La sintaxis de la definición de una función es la que se describió como estructura básica de una función.

3. Llamada o invocación de la función. Para que una función realice la tarea para la cual fue creada, debemos acceder o llamar a la misma. Cuando se llama a una función dentro de una expresión, el control del

programa se pasa a ésta y sólo regresa a la siguiente expresión de la que ha realizado la llamada cuando encuentra una instrucción return o, en su defecto, la llave de cierre al final de la función.

Generalmente, se suele llamar a las funciones desde la función main, lo que no implica que dentro de una función se pueda acceder a otra función.

Cuando queremos acceder a una función, debemos hacerlo mediante su nombre seguido de la lista de argumentos que utiliza dicha función encerrados entre paréntesis. En caso de que la función a la que se quiere acceder no utilice argumentos, se deben colocar los paréntesis vacíos.

La figura 6.1 nos muestra la posición que generalmente debe cumplirse cuando se hace uso de funciones.

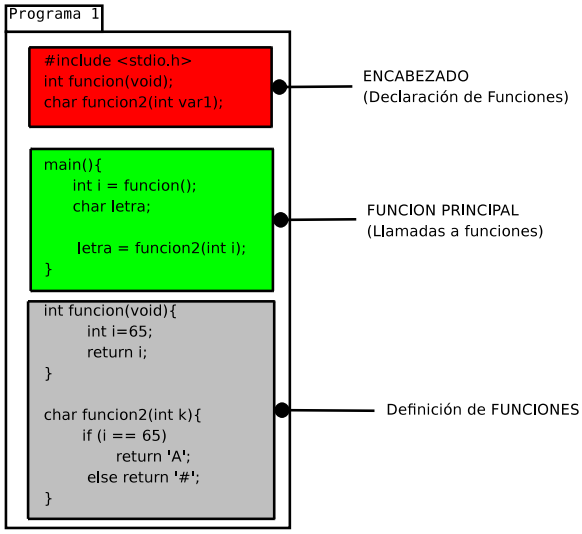

Figura 6.1: Descripción gráfica de las reglas para el uso de funciones en un programa.

Actividades

1. Hacer un programa que invoque o llame una función que suma dos números enteros y regresa el resultado como un tipo de dato `float`.

2. Hacer un programa que contenga la función sumatoria que recibe como argumentos el valor de inicio, fin e incremento de la sumatoria y resuelve la siguiente expresión:

$$H = \sum_{i=0,001}^{N} i^2 * e^-1$$

3. Hacer un programa que contenga la función prototipo `char aMayuscula(char letra)` y cada caracter que lea del flujo de entrada lo convierta a su equivalente en mayúscula, el programa debe terminar con el caracter `EOF`.

Práctica 7

Programación estructurada usando funciones Parte 2

Ahora veremos ejemplos de funciones en donde el paso de parámetros es por referencia. Para ello es necesario documentarse en relación a los operadores $*$ y `&`.

El operador $*$ se utiliza cuando se DECLARA o se DEFINE una función, mientras que el operador `&` se utiliza cuando se hace la llamada de la función.

Actividades

1. Hacer una función que no regresa ningún valor y recibe tres argumentos, los dos primeros valores que recibe la función se deben sumar y almacenar el resultado en el tercer argumento, el cual debe ser impreso despues de llamar la función en la función principal.

2. Hacer la función prototipo `void sumatoria(int inicio, int fin, int incremento, int *resultado);` que resuelve la sumatoria y el resultado es devuelto en la variable `resultado`.

Práctica 8

Recursividad.

8.1. Actividades

1. Hacer una función recursiva para obtener el resultado de una sumatoria.
2. Hacer una función recursiva que obtenga la longitud de una cadena de caracteres.
3. Determinar los valores de las variables durante los primeras 5 llamadas a la siguiente función recursiva.

```
#include <stdio.h>
int funcion(int a, int b); // Declaración de la función

main(){
        int a=100, b=4;

        int c = funcion(a,b);  // Llamada de la función
        printf("%d\n",c);
}
int funcion(int a, int b){  // Definición de la función
        if( a == 0)
            return b;
        else
            return funcion(a/b,b)*b;
}
```

Práctica 9

Arreglos

Los arreglos nos permiten definir un grupo de variables del mismo tipo. La estructura básica para definir un arreglo es el siguiente:

```
tipo_de_dato nombre[DIMENSION];
```

Un ejemplo de como se define un conjunto de 10 variables enteras es: `int arreglo[10];`. Una cadena de caracteres de longitud 50, se define como: `char cadena[50];`.

La figura 9.1 ilustra un arreglo de caracteres de dimensión 6 y en el que se encuentra almacenada la palabra `HOLA` y el indicador de fin de cadena en la posición 4 del arreglo, para el arreglo que define una cadena de caracteres. En la parte inferior de la figura 9.1 se define un arreglo de números reales.

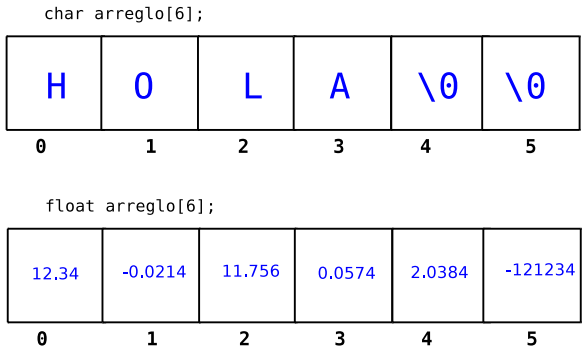

Figura 9.1: Descripción gráfica de un arreglo de dimensión 6 conteniendo la palabra `HOLA` en la parte superior y un arreglo de números reales en la parte inferior.

Actividades

1. Definir un arreglo de números enteros.
2. Definir un arreglo 100 números reales.
3. Definir una cadena de caracteres con longitud 200.
4. Que significa el caracter ’
   0’
5. Escribe a partir de la posición 10 de un arreglo la palabra `HOLA`.
6. Hacer una función que recibe dos parámetros, el primer parámetro es un arreglo de caracteres y el segundo es la posición en donde se debe insertar el caracter indicador de fin de cadena.
7. Escribir un programa que llene un arreglo de números reales y los imprima en orden inverso.

Práctica 10

Apuntadores

Un apuntador es una variable que contiene la dirección de otra variable. Se utilizan debido a que por lo general llevan un código más compacto y eficiente. Los apuntadores y los arreglos estan muy relacionados.Los apuntadores requieren disciplina, debido a que es muy fácil crear apuntadores que señalan a algún lugar inesperado.

Para entender los apuntadores es necesario tener una idea de la disposición de memoria de una computadora, la figura 10.1, nos muestra un ejemplo de las celdas y sus direcciones correspondientes a la memoria. De esa forma decimos que una dirección d ememoria puede almacenar un valor que puede corresponder a una dirección de memoria o se puede almacenar un valor asignado a una variable, tal y como es el caso del ejemplo mostrado en la figura 10.1.

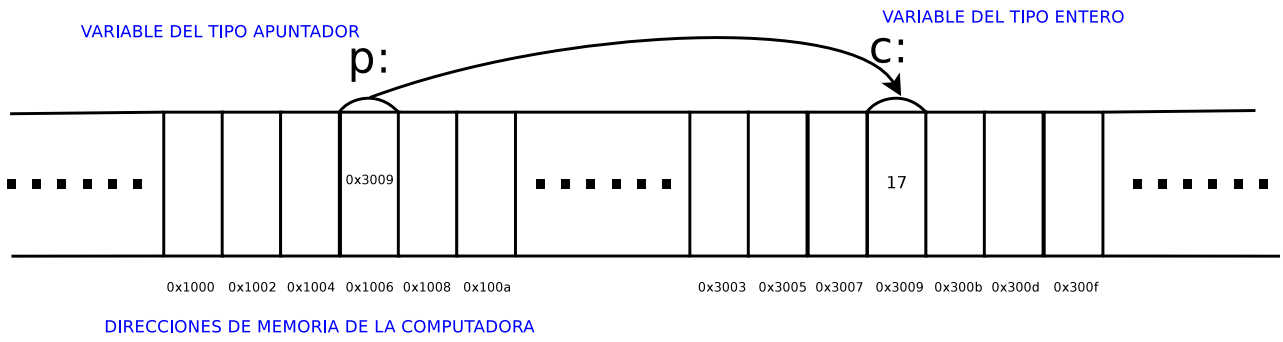

Figura 10.1: Disposición gráfica de la memoria, se ilustra que en las celdas se pueden almacenar direcciones de memoria (apuntador) y valores para variables de los tipos de datos descritos para el lenguaje (variable entera).

Actividades

1. Definir una variable entera y una variable del tipo entero apuntador o apuntador a enteros.
2. Haz que la variable de tipo apuntador apunte a la variable entera (se utiliza el operador unario &).
3. Asigna el valor de 17 a la variable entera que definistes.
4. Imprime en pantalla el contenido de lo que apunta la variable tipo apuntador (se utiliza el operador unario *).
5. Indica el significado de la siguiente expresión: $y = *ip + 1$. Recuerda que ip es una variable del tipo apuntador a entero.
6. Indica el significado de la expresión $*ip+ = 1$.
7. Indica el significado de las expresiones $++*ip$ y de $(*ip)++$.

8. Escribe la llamada correcta a la función `swap` que intercambia los valores entre dos variables que recibe como argumentos, la definición de la función es la siguiente:

```
void swap(int *px, int *py){
     int temp;

     temp = *px;
     *px = *py;
     *py = temp;
}
```

9. Imagina que se define un arreglo de enteros de la siguiente forma: `int A[10];`. Indica el significado para las siguientes expresiones:

   a) `int pa = &A[0];`

   b) `int x = *pa;`

   c) `int i=3; pa +i;`

   d) `(pa + 1);`

10. Define un arreglo de apuntadores y proporciona un ejemplo de uso.

Práctica 11

Uso de cadenas de texto

Palabras del lenguaje por conocer en el tema:

- tolower()
- toupper()
- strlen()
- isdigit()
- isupper()
- isalpha()

11.1. Actividades

1.

Práctica 12

Estructuras y Uniones.

Palabras del lenguaje por conocer en el tema:

1. struct
2. typedef

Las estructuras nos permiten agrupar un conjunto de variables de tipos de datos diferentes. Se pueden encapsular variables de diferentes tipos de datos en un objeto como lo muestra la figura 12.1.

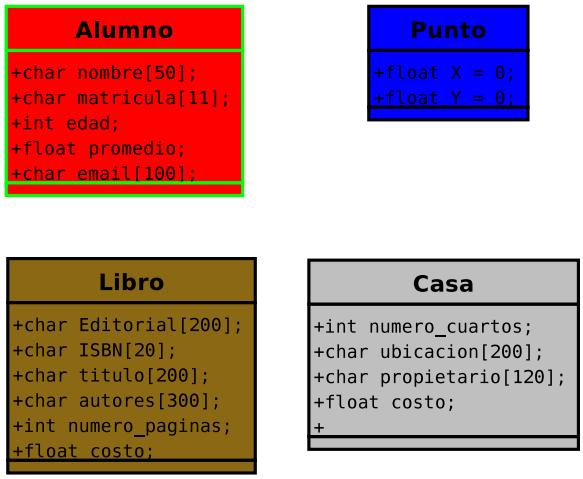

Figura 12.1: Una estructura puede representar objetos agrupando las diferentes variables que describen el objeto.

Actividades

1. redefine el tipo de dato `int` a `entero`.
2. Defina una estructura punto, que contenga dos valores enteros: x, y.
3. Defina una estructura rectángulo, que contenga dos puntos y un caracter que indica la linea del rectángulo.
4. Crear un programa que reciba dos puntos y un caracter y se los asigne al objeto rectángulo.
5. Crear una función que imprima el rectángulo en pantalla, considerando el prompt del sistema como la posición (0,0).
6. Crear una función para escalar el rectángulo, los parámetros de la función seran el rectángulo a escalar y el factor de escalamiento, por ejemplo la deficnición de la función puede ser:

```
void escalar(struct rectangulo R, int factor_escala);
```

7. Agregarle un tercer elemento a la estructura rectángulo que sea una matriz de caracteres, en la cual se almacene el dibujo del rectángulo.

Práctica 13

Lectura y/o Escritura de archivos Parte I

Palabras del lenguaje por conocer en el tema:

- FILE *
- fopen
- fclose
- feof
- fgetc, fputc

EL símbolo que representa interactuar con un archivo de datos en un diagrama de flujo es el que se muestra en la figura 13.1.

Figura 13.1: Figura que representa transacciones con un archivo de datos.

La estructura básica para el manejo de archivos se presenta en el siguiente programa:

```
#include<stdio.h>

main(){

      FILE *apArchivo;  //1o. se define la variable del tipo ARCHIVO.

      apArchivo = fopen("datos.txt","rw");  // abrimos el archivo en modo lectura y escritura.

      // utilizamos funciones que realizan operaciones sobre los archivos
      // como fscanf, fgetc para leer datos y fprintf, fputc para escribir en el archivo.

      fclose(apArchivo);  // cerramos el apuntador al archivo.
}
```

Actividades

1. Hacer un programa que lea caracter por caracter desde un archivo y los imprima en pantalla.

2. Hacer un programa que lea los datos de un archivo y nos indique cuantos renglones tiene.

3. Hacer una función que recibe como parámetro el nombre de un archivo de datos que contiene 100 números reales, la función lee los números y obtiene la sumatoria de los números. Finalmente la función regresa el resultado de la sumatoria. Crear el diagrama de flujo para el programa.

4. Hacer un programa que escribe en un archivo que recibe como parámetro todo el texto que se escribe por

Práctica 14

Lectura y/o Escritura de archivos Parte II

Palabras del lenguaje por conocer en el tema:

14.1. Actividades

Planteamiento de un proyecto:

La empresa de noticias ?FIE Informate?, le solicita información estadística de un conjunto de archivos de noticias que tiene en diferentes formatos. Para ello se le pide que desarrolle una aplicación en el lenguaje C, la cual debe cumplir con los siguientes requerimientos:

1. Debe clasificar los archivos según la extensión de que se trate: TXT, DOC, PDF, JPG, GIF, etcétera, mostrar un menú que le permita al usuario listar los archivos según la extensión y ordenados por el tamaño de menor a mayor. El programa lista alfabéticamente todas las extensiones existentes en el directorio que se almacenan los archivos.

2. Para los archivos de texto debe guardar un registro con los siguientes datos Nombre del archivo Tamaño Fecha de creación Numero de palabras diferentes contenidas en el texto. Palabra más frecuente y menos frecuente en el archivo. Frecuencia de cada una de las palabras diferentes. Numero de palabras en el texto eliminando las palabras del archivo stopwords.txt o también conocidas como palabras vacías.

3. Crear un diccionario de palabras de todos los archivos de texto contenidos en el directorio y guardar el diccionario en un archivo llamado dicc.txt. Mostrar el numero de palabras del diccionario y presentar en pantalla el diccionario.

4. Debe permitir realizar búsquedas por palabra en en diccionario, es decir, verificar si una palabra existe o no en el diccionario.

5. Mostrar la palabra más frecuente, la menos frecuente, el promedio de las palabras y la desviación estándar del conjunto de archivos de texto.