Licenciatura en Sistemas
Programación de computadoras

Equipo docente: Jorge Golfieri, Natalia Romero, Romina Masilla y Nicolás Perez
Mails: jgolfieri@hotmail.com , nataliab_romero@yahoo.com.ar , romina.e.mansilla@gmail.com, nperez_dcao_smn@outlook.com
Facebook: https://www.facebook.com/groups/171510736842353
Git: ***http://***github.com/UNLASistemasProgramacion/Programacion-de-Computadoras

---

Unidad 3:

Estructuras de control: iteración (for, while, do while). Problemas: implementación y testeo. Algoritmos de búsqueda: secuencial y binaria. Entrada / Salida desde consola. Nociones de recursividad. Problemas: implementación y testeo.

Bibliografía citada:

Fundamentos de programación: libro de problemas. Algoritmos, estructuras de datos y objetos (2a. ed.) Luis Joyanes Aguilar, Luis Rodríguez Baena y Matilde Fernández Azuela. McGraw-Hill España. Capítulo 5-14.

C algoritmos, programación y estructuras de datos. Luis Joyanes Aguilar, Andrés Castillo Sanz, y Lucas Sánchez García. Capítulo 6-8.

Unidad 3: For – while – do while

Por ahora, hemos visto estructuras de condiciones, las ya nombras **IF** y **SWITCH CASE**; ambas son básicamente lo mismo, simplemente optamos por utilizar un SWITCH CASE en el caso que tengamos varias condiciones semejantes y no hay una lógica muy compleja en cada una de estas condiciones. Cabe descartar y repetir, que esto solo lograra mejorar la eficiencia del programa, pero no nos dará muchos mas beneficios, **uno puede optar por saber solo uno de estos dos métodos y será tan buen programador como alguien que sabe ambos**, pero es bueno conocerlos y aunque sea compararlos alguna vez.

A modo de recuerdo, observemos estos dos ejemplos, con IF y con SWITCH CASE.

Ejemplo 1:

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

int main(void) {

	int a;
	switch(a)
	{
	case 0: //operaciones a realizar en caso de que a valga 0;
	 break;
	case 1://mismo proceso
```

```
        break;
        ...
        case n:  //n tiene que ser una constante, o numérica {0,1,2 ....} o una definida
#define n
        break;
        default:  //en caso de que no se de ninguna de las anteriores hacer...
}
return 0;
}
```

Ejemplo 2: Hagamos lo mismo con la estructura del IF

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

int main(void) {

        int a;
        if(a==0)  //operaciones para 0
        else if(a==1) //operaciones para 1
        ...
        else if(a==n)   //operaciones para n
        else //cualquier otro caso
return 0;
```

```
}
```

Pero al finalizar la semana pasada vimos otro tipo de estructura que es notoriamente distinta al IF y SWITCH. Hemos comenzado a hablar del ***FOR***.

El **ciclo For** es una de las instrucciones más sencillas de aprender, y consiste en utilizar mayormente "rangos" en los cuales se define el número de iteraciones que se pueden hacer en este ciclo. Es la estructura madre de las repeticiones de acciones. Si uno quiere realizar una misma acción varias veces, el FOR es la opción indicada.

La sintaxis es la siguiente:

```
1 for(inicio;mientras;incremento)
2 {
3  //CODIGO A EJECUTAR
4 }
```

Donde el inicio es la declaración de una variable que funciona como un "contador" mientras ejecutamos el ciclo. Continuamente el "mientras" especifica los valores o el rango de valores que puede tomar el contador de "inicio" y el "incremento" especifico cuanto se va a incrementar el contador con cada iteración, lo que indicaría que eventualmente el contador saldría de su posible rango de valores y terminaría el ciclo.

Ejemplo 3: Mostrar solo los números pares, del 0 al 20.

```
#include <stdio.h>


int main(int argc, const char * argv[])
{ //CICLOS FOR EN C
        int x;
```

```
        for (int x=2;x<20;x+=2) {
        printf("El contador X vale: %d \n", x);
    }
   return 0;
}
```

Ejemplo 4: Escriba un programa en lenguaje C que solicite el ingreso de dos números (valor inicial y un valor final) y que muestre por pantalla los números que pertenecen al intervalo dado.

```
#include <stdio.h>
void main()
{
        //Declaración de variables
        int inicial, final, i;
        //Solicitando y leyendo el ingreso de datos desde el teclado
        printf("Ingrese el valor inicial: ");
        scanf("%d",&inicial);
        printf("Ingrese el valor final: ");
        scanf("%d",&final);
                for(i=inicial   ;i<=final;  i++)
        {
```

```
                printf("%d\n",i); //mostrando por pantalla los números desde

                                  //el valor inicial hasta el valor final

        }

}
```

Otra estructura fundamental para las sentencias con repeticiones es el WHILE (mientras).

- Este ciclo puede o no ejecutar el bloque de instrucciones que contiene, ya que antes de comenzarlo, evalúa una condición, si esta condición no se cumple, el ciclo jamás comenzara.

- En caso de que se cumpla la condición, el ciclo comenzara y cuando termine de ejecutar el bloque de instrucciones volverá a evaluar la condición, esto permite delimitar cuantas veces se repetirá el ciclo.

Por ejemplo, si la condición que evaluara WHILE es que una variable CONT inicializada a 1 sea menor a 11 y dentro del bloque instrucciones del WHILE sumamos 1 a CONT, cada vez que se repita el ciclo se irá incrementando en 1 el valor que contenga la variable CONT, cuando llegue a 11, la condición del WHILE ya no se cumplirá y el ciclo ya no se repetirá, con esto sabemos que el ciclo se repetirá 10 veces.

Ejemplo 5: El clásico ejercicio de sumar los 20 primeros números, pero utilizando el WHILE.

```
#include<stdio.h>

#include<stdlib.h>

void main()
```

```
{
        int num=0;
        int suma=0;

        while(num<=10)  {
                suma = suma + num;
                num++;
};
        printf(“Suma = %d\n”,suma);
        system(“pause”);
}
```

Ejemplo 6: Que nos diga si cada número que tecleemos es positivo o negativo, y que pare cuando tecleemos el número 0.

```
#include <stdio.h>

int main()
{
        int numero;

        printf("Teclea un número (0 para salir): ");
        scanf("%d", &numero);
```

```
        while (numero!=0)
                {
                        if (numero > 0) printf("Es positivo\n");
                         else printf("Es negativo\n");
                        printf("Teclea otro número (0 para salir): ");
                        scanf("%d", &numero);
  }
  return 0;
}
```

Para terminar, veremos el DO…WHILE, hacer mientras, es semejante al WHILE, solo que la condición se verifica al final, esto nos permite ingresar SI O SI una vez al bucle.

Al igual que en el caso anterior, si queremos que se repitan varias órdenes (es lo habitual), deberemos encerrarlas entre llaves. Nuevamente, puede ser recomendable incluir siempre las llaves, como costumbre.

Ejemplo 7: vamos a ver cómo sería el típico programa que nos pide una clave de acceso y nos deja entrar hasta que tecleemos la clave correcta. Eso sí, como todavía no sabemos manejar cadenas de texto, la clave será un número:

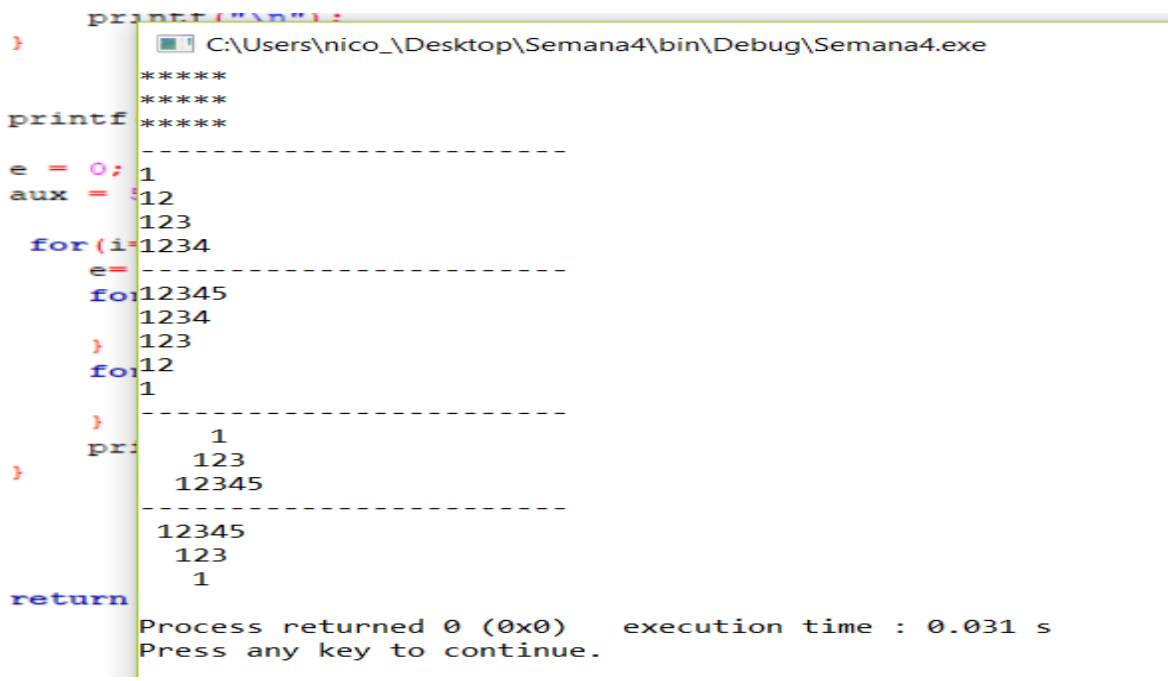
```
#include <stdio.h>


int main()
{
        int valida = 711;
        int clave;
```

```
        do
        {
         printf("Introduzca su clave numérica: ");
        scanf("%d", &clave);
        if (clave != valida) printf("No válida!\n");
  }
  while (clave != valida);
  printf("Aceptada.\n");


  return 0;
}
```

Supongamos que queremos dibujar en pantalla lo siguiente:



Es decir, un rectángulo, triangulo, triangulo invertido, pirámide, invertida, etc. **¿Como lo harian utilizando las estructuras de control que hemos trabajado?**

Los invitamos a que todos logren mostrar por pantalla esos resultados. A continuación, se adjunta una posible solución a estos problemas para que la comparen con la solución que han tomado ustedes:

```
#include <stdio.h>

#include <stdlib.h>


int main()

{
```

```
    //ALGORITMO: Dibujar rectangulo de 3 x 5
    //i y j para recorrer los renglones y las filas, 3 x 5
    int i = 0;
    int j = 0;

    //Recorremos lso renglones del 1 al 3
    for(i=1;i<=3;i++){
            //por caada Renglon dibujamos 5 *
        for(j=1;j<=5;j++){
            printf("*");
        }
        printf("\n");
    }

    printf("-----------------------\n");

    //Triangulo
    //+
    //++
    //+++
    //++++
```

```
    //Algortimo: Dibujar un triangulo de 4 renglones de 1 a 4
    for(i=1;i<=4;i++){
        for(j=1;j<=i;j++){
            printf("%d",j);
        }
        printf("\n");
    }

    printf("----------------------\n");

    //Triangulo invertido
    for(i=5;i>=1;i--){
        for(j=1;j<=i;j++){
            printf("%d",j);
        }
        printf("\n");
    }

    printf("----------------------\n");

    //Piramide
    //__x
```

```
    //_xxx
    //xxxxx
    int e = 5;
    int aux = 0;

    for(i=1;i<=5;i+=2){
        e= e-1;
        for(aux=1; aux<=e;aux++){
                printf(" ");
        }
        for(j=1;j<=i;j++){
            printf("%d",j);
        }
        printf("\n");
    }

    printf("-----------------------\n");

    e = 0;
    aux = 5;

     for(i=5;i>=1;i-=2){
```

```
        e= e+1;
        for(aux=1; aux<=e;aux++){
                printf(" ");
        }
        for(j=1;j<=i;j++){
            printf("%d",j);
        }
        printf("\n");
    }


    return 0;
}
```

Sigamos mejorando este proyecto. ¿Qué creen que podríamos mejorar? Estamos recorriendo los FOR con índices denominados “i” y “j”, ¿Que tan bien creen que esta eso?, si bien es muy frecuente y habitual realizar los recorridos con estos índices i, j,k,l,m,n, les proponemos que de ahora en adelante les pongan **nombres mucho más descriptivos** de lo que en verdad se esta recorriendo, por ejemplo, en nuestros casos, i llamarlo renglón, y j columna.

De esta forma el código se tornará mucho mas legible por cualquier persona. Además, les proponemos que **comenten todas las líneas de código**, aun cuando piensen que

saben a la perfección lo que realizan en cada línea de comandos, crean que puede parecer muy molesto e inútil, pero les simplificara su vida profesional en no mucho tiempo.

Pruebas de escritorio:

Escribir en el pizarrón seguramente sea muy tedioso y aburrido para ustedes, así como complicado para nosotros, quizás cometemos algún **error de sintaxis** por no tener una aplicación como code-blocks que nos indica el error, hasta incluso podremos tener algún **error lógico** puesto que no podemos compilar y probar nuestros códigos. Además, puede que a ustedes les resulte muy complicado seguirnos o entender parte de nuestros razonamientos plasmados en el pizarrón en un lenguaje de programación. Por eso es por lo que existen **las pruebas de escritorio.**

La prueba de escritorio no es más que efectuar un proceso de simulación con el algoritmo desarrollado (ver que haría la computadora). Este trabajo se realiza en base a una tabla cuyos encabezados son las variables que se usan en el algoritmo y debajo de cada una de ellas se van colocando los valores que van tomando, paso a paso y siguiendo el flujo indicado por el algoritmo, hasta llegar al final.

Es la etapa más importante en el desarrollo de un programa, por cuanto el realizar la prueba de escritorio nos permite saber:

1. Si el programa hace lo que debería hacer
2. Si no hace lo que debería hacer, nos permitirá detectar errores como ser:
    - Si algún paso o instrucción no está en el orden correcto
    - Si falta algo
    - Si algo esta demás
    - Si los pasos o instrucciones que se repiten lo hacen más o menos veces de lo debido
    - Si las instrucciones están en un orden apropiado
    - Otros errores que pueden presentarse
3. Elegir los datos apropiados para la prueba

La prueba consistirá en 2 etapas:

- La primera, en probar inicialmente que el programa funcione correctamente, para lo que se elegir´ algunos datos fáciles de probar, cosa que siempre es posible.
- La segunda, si se prueba que ya funciona, se buscarán otros datos (si los hay) que hagan que falle el algoritmo, en cuyo caso se habrán de detectar otros errores. Si el algoritmo no falla, podemos concluir que el programa está terminado y revisado, por lo tanto, correcto.

Para hacer un ejemplo de prueba de escritorio, usemos una ejercitación simple para repasar lo visto y entender que es una prueba de escritorio.

Ejercicio prueba de escritorio: Realizar un algoritmo que pida el ingreso de 3 números enteros, mostrar por pantalla el mas chico y el mas grande. Mostrar por pantalla todos los números entre el mas chico y el mas grande.

Una propuesta de solución al problema sería:

```
//Iniciamos las variables

    int numero1 = 0;

    int numero2 = 0;

    int numero3 = 0;

    //Para encontrar maximos y minimos suelen definirse numeros muy grandes y muy chicos

    int numeroMinimo = 999999;

    int numeroMaximo = -999999;
```

```
//Los pedimos y guardamos por pantalla
printf("Dame el primer numero: \n");
scanf("%d", &numero1);
printf("Dame el segundo numero: \n");
scanf("%d", &numero2);
printf("Dame el tercer numero: \n");
scanf("%d", &numero3);


//busquemos el minimo
if (numero1<=numeroMinimo){
    numeroMinimo = numero1;
};
if (numero2<=numeroMinimo){
    numeroMinimo = numero2;
};
if (numero3<=numeroMinimo){
    numeroMinimo = numero3;
};
//buscamos el maximo
if (numero1>=numeroMaximo){
```

```
    numeroMaximo = numero1;
};
if (numero2>=numeroMaximo){
    numeroMaximo = numero2;
};
if (numero3>=numeroMaximo){
    numeroMaximo = numero3;
};

//Mostramos los numeros entre el mas chico y el mas grande
int indice;
for (indice = numeroMinimo; indice<=numeroMaximo ; indice++){
    printf("%d  ", indice);
}
```

Hagamos ahora la prueba de escritorio de dicho código:

Instrucciones	***i***	***numero1***	***numero2***	***numero3***	***min***	***max***	***Consola:***
Inicio de variables		**0**	**0**	**0**	**99999**	**-99999**	
printf y scanf del numero1		**-3**	0	0	99999	-99999	
printf y scanf del numero2		-3	**6**	0	99999	-99999	
printf y scanf del numero3		-3	6	**2**	99999	-99999	
if con numero1 <= min		-3	6	2	**-3**	-99999	
if con numero2 <= min		-3	6	2	-3	-99999	
if con numero3 <= min		-3	6	2	-3	-99999	
if con numero1 >= max		-3	6	2	-3	**-3**	
if con numero2 >= max		-3	6	2	-3	**6**	
if con numero3 >= max		-3	6	2	-3	6	
inicio el indice i	**0**	-3	6	2	-3	6	
for desde i= min hasta max	**-3**	-3	6	2	-3	6	**"-3"**
	-2	-3	6	2	-3	6	**"-3, -2"**
	-1	-3	6	2	-3	6	**"-3,-2,-1"**
	0	-3	6	2	-3	6	**"-3,-2,-1,0"**
		-3	6	2	-3	6	
fin for	**6**	-3	6	2	-3	6	**"-3,-2,-1,0,1,2,3,4,5,6"**

Vemos como la prueba de escritorio nos va mostrando, paso a paso, como va cambiando de valor cada variable a medida que vamos incursionando de forma ordenada en el código que construimos. Además, nos muestra de una forma muy tangible cómo será el resultado que se mostrará por la consola, para compararlo cualitativamente con lo que esperábamos del software escrito.

Ejemplo: Hagamos una diagonal de +, queremos que nos quede algo de este estilo:

+

 +

  +

   +

    +

En el primer renglón no dejamos espacios, en el 2do renglón dejamos un espacio, en el 3er renglón dejamos 2 espacios y asi hasta llegar al 5to renglón donde dejaremos 4 espacios. Hagan la prueba de escritorio de dicho programa.

Solución con For:

```
printf("Recordemos un poco de for con un ejemplo simple: \n");
int renglon = 0;
int columna = 0;

//Son 5 renglones asi que vamos desde el 1 al 5, para los que no
//les gusta arrancar del 0 :(

for (renglon = 1; renglon<=5;renglon++){

    //Cuando el renglon es el 1 no hay que dejar espacios, por eso
    //el for va de 1 a 1 sin incluir el 1, o sea no hace nada
    //cuando el renglon es el 2, el for va de 1 a 2, o sea entra una
    //sola vez y pone un espacio
    //luego de poner los espacios se pone el + y se deja un renglon
```

```
        for(columna = 1; columna<renglon;columna++){
            printf(" ");
        }
        printf("+ \n");

    }
```

Solución con While:

```
    int renglon = 1;
    int columna = 1;

    //Nuestro ejercicio es el mismo, solo que realizaremos la
    //escritura en pantalla MIENTRAS el renglon sea menor o igual a 5
    while(renglon<=5){// voy del renglon uno al 5
            columna = 1; //inicio la columna 1 del renglon
            while(columna<renglon){//hago un espacio menos que el renglon en el
```

```
                //que estoy parado

            printf(" ");

            columna = columna + 1; //esto juega el roll del i++ del For

        }

        printf("+ \n"); //pongo el + y dejo renglon

        renglon = renglon + 1; //Cambio de renglón

}
```

¿Qué diferencias vemos entre el for y el while?: Ambas son para realizar actividades repetitivas, solo que en el for sabemos la dependencia que hay entre el inicio y el fin de la actividad, mientras que en el while solo nos interesa saber en qué momentos queremos que deje de realizarse la tarea.

A priori ambas sirven para realizar las mismas acciones en un lenguaje de programación, solo que como en el while no hay que especificar el inicio y el fin, es decir no hay que poner de forma explicita la cantidad de repeticiones, hay que tener MUCHO mas cuidado con los incrementos de las variables que dan el inicio y el fin, cosa que en el for se podía descuidar mucho más, ya que uno definía el inicio, fin y los saltos desde el primer renglón.

Por ejemplo, en las líneas de código arriba realizadas, vemos una clara comparación entre el while y el for, no difieren mucho. Pero como el while no posee, el “renglón= 1”, renglón ++, hay que manejar esa variable y su salto dentro del mismo while.

Practica Unidad 3:

a) Crear un programa que pida al usuario su contraseña (numérica). Deberá terminar cuando introduzca como contraseña el número 4567, pero volvérsela a pedir tantas veces como sea necesario a menos que ingrese un numero negativo.

b) Crea un programa que escriba en pantalla los números del 1 al 10.

c) Crea un programa que escriba en pantalla los números pares del 26 al 10.

d) Crear un programa calcule cuantas cifras tiene un número entero positivo (pista: se puede hacer dividiendo varias veces entre 10).

e) Crear un programa que pida números positivos al usuario, y vaya calculando la suma de todos ellos (terminará cuando se teclea un número negativo o cero o cuando la suma supere los 1500).

f) Crear un programa que muestre los números del 15 al 5, descendiendo (pista: en cada pasada habrá que descontar 1, por ejemplo, haciendo i--).

g) Crear un programa que muestre los primeros ocho números pares (pista: en cada pasada habrá que aumentar de 2 en 2, o bien mostrar el doble del valor que hace de contador).

h) Crear un programa que escriba en pantalla la tabla de multiplicar del 6.

i) Crear un programa que escriba en pantalla los números del 1 al 50 que sean múltiplos de 3 (pista: habrá que recorrer todos esos números y ver si el resto de la división entre 3 resulta 0).