Resolución de Problemas y Algoritmos

Clase 10
Resolución de Problemas

Grace Murray Hopper

Dr. Alejandro J. García
http://cs.uns.edu.ar/~ajg

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca - Argentina

Aviso importante: CAMBIO de AULA y HORA primer parcial

Primer parcial: jueves 5 de mayo de 14:00 a 18:00 (horario distinto al de clase)
En un aula nueva: aula 8 de Palihue (edificio rosa). Si no conoce el aula consultar con los docentes en alguna clase antes día del examen. (ver en Google maps)

Conceptos: diferentes clases de errores en programas

- **Error de compilación:** es un error detectado por el compilador al momento que se está realizando la compilación de un código fuente, por eso también se llama *error en tiempo de compilación*.
- **Error de ejecución:** ocurre cuando al momento de la ejecución del programa hay una situación anormal, y generalmente provoca que la ejecución se corte abruptamente. También se lo llama *error en tiempo de ejecución*. Ejemplo: intentar leer de un archivo vacío.
- **Error lógico:** también llamado **error de programación** (*bug*), es un error en la lógica del algoritmo o programa el cual causa que no se resuelva correctamente la tarea que debe hacer el programa.

Un profesional debe lograr que los programas no tengan errores.
Para ello *deben probarse* los *programas* (en Inglés *testing*) con los suficientes casos de prueba.
Si se detecta un mal funcionamiento con algún caso de prueba, entonces se deben *buscar y eliminar* los *errores* (*debugging*).

Conceptos: debugging (depuración)

Debugging (depuración): se refiere al proceso metodológico de buscar y reducir el número de errores o defectos (*bugs*) de un programa, con el objetivo de lograr que el programa tenga el comportamiento esperado.

http://en.wikipedia.org/wiki/Debugging

El término "debugging" es atribuido a la Almirante Grace Murray Hopper.

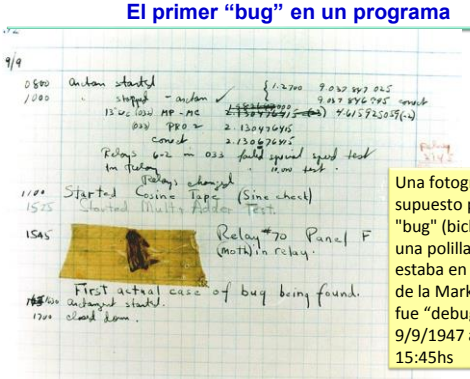
En 1947 mientras trabajaba en una Mark II encontró una polilla atrapada en un relé que impedía las operaciones de dicha computadora.

Grace dijo que al sacar el "bug" (bicho) habían hecho un "debugging" (desbichado) del sistema.

En una entrevista Grace comentó que el término "bug" como sinónimo de error ya era usado en la jerga (ver más sobre esto en software-bug).

Grace Murray Hopper frente al teclado de UNIVAC I (en 1960)

El primer "bug" en un programa



Una fotografía del supuesto primer "bug" (bicho) real: una polilla (moth) que estaba en el relay#70 de la Mark II. La cual fue "debugged" el 9/9/1947 a las 15:45hs

Ejemplo propuesto

Problema: escriba un programa que busque el menor elemento en un archivo de enteros "numeros.enteros" el cual ya fue creado.

Ejemplos significativos / casos de prueba
Archivo = 1, 2, 3 Archivo = 8, 3, 4, 3, 4
Archivo = vacío Archivo = 8
Archivo = 3, 2, 1

Algoritmo buscar el menor
Abrir el archivo para leer
Leer el primer elemento
Guardar el primero como el menor encontrado hasta ahora
Repetir mientras no sea fin de archivo (EOF)
  Leer un elemento nuevo del archivo
  Si el elemento recién leído es menor al menor encontrado hasta ahora
    entonces el elemento recién leído es el nuevo menor encontrado
Mostrar el elemento menor
Cerrar el archivo.
fin.

Realizaremos el programa en el pizarrón

Parte de la implementación que hicimos en el pizarrón

```
Program buscar_menor;
TYPE mis_nums = FILE OF integer; // tipo nuevo creado por el constructor
VAR numeros: mis_nums; // manejador del archivo de enteros
    menor, leido : integer; // el menor encontrado y el último leído
begin
assign(numeros, 'numeros.enteros'); // vincula nombre con el manejador
reset(numeros); // abre el archivo para leer
IF eof(numeros) THEN writeln( ' archivo vacío, no hay menor' )
ELSE BEGIN
    read(números,menor); //guardo el primero como el menor encontrado
     // ahora miro el resto del archivo para ver si hay otro menor….
    WHILE NOT eof(numeros) DO // mientras no sea el fin del archivo
      begin

        … // el resto del programa fue desarrollado en el pizarrón
```


Tarea propuesta

Problema: escriba un programa que indique cual es y cuantas veces aparece el menor elemento en un archivo de enteros "numeros.enteros" el cual ya fue creado.

Ejemplos significativos / casos de prueba

Archivo = -9, 2, 3, -9, 2, -9	el menor es el -9 y está tres veces
Archivo = 8, 8, 4, 3, 4, 3	el menor es el 3 y está dos veces
Archivo = vacío	no existe el menor
Archivo = 8	el menor es el 8 y está una vez
Archivo = 3, 3, 3, 1	el menor es el 1 y está una vez

Algoritmo cuantas veces el menor

Tarea para practicar

(no deje de mostrarle a un docente su propuesta)

fin.

Problema: escriba un programa para copiar el contenido de "mis-numeros.datos", a otro archivo "mis-numeros.copia".

```
Program incorrecto; {VERSION INCORRECTA}
TYPE archi_enteros = FILE OF integer;
VAR F1,F2: archi_enteros;
begin
    assign(F1, 'mis-numeros.datos');
    assign(F2, 'mis-numeros.copia');

    F1:=F2;

    close(F2); close(F1);
end.
```

MAL

Esto no copia los archivos

Esta asignación hace que los dos manejadores se refieran al mismo archivo: "mis-numeros.copia"

Problema: escriba un programa para copiar el contenido de "mis-numeros.datos", a otro archivo "mis-numeros.copia".

```
Program copiar; {version correcta}
TYPE archi_enteros = FILE OF integer;
VAR F1,F2: archi_enteros ;  elemento: integer;
begin
    assign(F1, 'mis-numeros.datos');  reset(F1);  // F1 para lectura
    assign(F2, 'mis-numeros.copia');  rewrite(F2); // F2 para escritura
    while not eof(F1) do begin // mientras no se acabe F1
     read(F1,elemento); // leo de a un elemento de F1
     write(F2, elemento); // escribo cada elemento leído en F2
    end;
    close(F1); close(F2);
end.
```


Eliminar un elemento de un archivo

Problema: escriba un programa que solicite al usuario un elemento y elimine todas las apariciones de ese elemento del archivo de enteros "numeros.enteros"

Ejemplos significativos / casos de prueba

Archivo = 1, 2, 3 eliminar el 2	Archivo = 1, 3
Archivo = 1, 2, 3 eliminar el 5	Archivo = 1, 2, 3
Archivo = 8, 3, 4, 3, 4 eliminar el 4	Archivo = 8, 3, 3
Archivo = vacío eliminar el 2	Archivo = vacío
Archivo = 8 eliminar el 8	Archivo = vacío

Realizaremos el algoritmo y programa en el pizarrón

Eliminar un elemento de un archivo

Problema: escriba un programa que solicite al usuario un elemento y elimine todas las apariciones de ese elemento del archivo de enteros "numeros.enteros"

Realizaremos el programa en el pizarrón

Algoritmo eliminar un elemento

Pedir y leer el elemento a eliminar

Abrir el archivo original para leer, y abrir un archivo auxiliar para escribir

mientras no sea fin de archivo original

  Leer un elemento del original

  Si el elemento recién leído es distinto al que quiero eliminar

    entonces escribir el recién leído al archivo auxiliar

Cerrar ambos archivos

Abrir auxiliar para leer y original para escribir

Copiar todos los elementos del auxiliar al original.

Como fue explicado en el problema anterior

fin.

Problema propuesto

Problema: **Escribir un programa para determinar si un número natural N es primo**

Definición: un número es primo si es un entero positivo mayor que 1, que es divisible solamente por si mismo y la unidad.

Ejemplos (futuros casos de pruba):

2 , 3, 7, 11 y 2003: son primos
4 no es primo es divisible por 2
2001 no es primo, es divisible por 3
121 no es primo, es divisible por 11

Una solución: **si N tiene un divisor distinto de N o 1 entonces NO es primo, de lo contrario es primo. Entonces para hacer un algoritmo puedo asumir que es primo hasta que se demuestre lo contrario (esto es, voy viendo una a uno si encuentro un divisor)**

Algoritmo para "esPrimo"

```
ALGORITMO EsPrimo
COMIENZO
 leer el número N
 SI (N > 1)
 ENTONCES EsPrimo ← Verdadero {asumo que es primo}
 SINO EsPrimo ← Falso
 Divisor ← 2 { el primer divisor a considerar es 2 ... }
 REPETIR MIENTRAS (Divisor < N) y (esPrimo = verdadero)
     SI N es divisible por Divisor ( usando módulo: N// Divisor = 0)
         ENTONCES esPrimo ← falso {Si Divisor divide a N, NO ES PRIMO}
     Divisor ← Divisor + 1 {paso al próximo posible Divisor }
 fin repetir mientras
FIN ALGORITMO
```

Tarea 1: Implemente en Pascal este algoritmo usando WHILE.
Tarea 2: Luego implemente usando REPEAT-UNTIL