RESOLUCIÓN DE PROBLEMAS Y ALGORITMOS

CLASE 14

Funciones definidas por los programadores.

Luciano H. Tamargo
http://cs.uns.edu.ar/~lt
Depto. de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur, Bahía Blanca
2016

CONSULTA

- CAMBIA DE HORARIO LA CONSULTA DEL PROFESOR!
  - DESDE LA SEMANA DEL 10/10
  - SERÁ LOS **VIERNES A LAS 17HS**
  - EN EL NUEVO EDIFICIO DEL DCIC.

SOBRE EL TRABAJO PROFESIONAL FUTURO

- Como profesional, un informático debe tener la capacidad de resolver problemas utilizando computadoras.
- Estos problemas pueden ser de:
  - ✓ **muy gran escala**: *como por ejemplo mantener en órbita a la Estación Espacial Internacional (ISS).*
  - ✓ **gran escala**: por ejemplo desarrollar un sistema de reserva y venta de pasajes de una compañía aérea.
  - ✓ **pequeña escala**: *como por ejemplo validar la identidad de un usuario mediante su nombre y clave; o controlar que una fecha ingresada en un formulario de una página web sea correcta.*

REFLEXIONE UN MOMENTO…

¿Cómo será el trabajo profesional si hay que desarrollar software a gran escala?
¿Trabajaré en grupo?
¿Tendré a cargo una parte?
¿Me relacionaré con otros?
¿Tardaré varios días en hacer parte de mi tarea?

- Los lenguajes de programación evolucionaron para permitir que los programadores puedan **hacer sus propias primitivas**.
- De esta forma pueden **compartir** y "**re-utilizar**" el código lo más posible y de la mejor manera posible.

MOTIVACIÓN

- Indique a que clase de elemento de Pascal corresponden los ejemplos de cada columna:

then	**integer**	**TRUNC**	**WRITE**
and	**real**	**EOF**	**READ**
program	**text**	**CHR**	**RESET**

- Agregue un ejemplo más a cada columna de la tabla.
- ¿En qué lugar de un programa en Pascal se pueden utilizar los elementos de la tercera y cuarta columna?

FUNCIONES Y PROCEDIMIENTOS PREDEFINIDOS

- Al programar en Pascal han usado **primitivas predefinidas**. Por ejemplo:

```
…
ASSIGN(F1,'mi-archivo');
WRITELN('ingrese un número');
READLN(num);
RESET(F1);
REWRITE(F2);
while not EOF(F1) do
  begin
    READ(f1,elem);
    if TRUNC(elem) > SQR(num) then
      WRITE(F2, elem)
    else
      WRITELN(TRUNC(elem),'menor que',SQR(num));
  end;
```

Primitivas como sentencias (**procedimientos predefinidos**)

Primitivas en expresiones (**funciones predefinidos**)

¿Quién creó el código de estas primitivas predefinidas?

FUNCIONES EN PASCAL

- Ejemplos de algunas **funciones predefinidas** :
  - **EOF**(F): recibe un manejador y retorna boolean
  - **TRUNC**(R): recibe real y retorna integer
  - **SQRT**(R): recibe real y retorna real
  - **CHR**(I): recibe integer y retorna char
- Reflexionemos sobre las Funciones en Pascal:
  - Se utilizan en una expresión.
  - Reciben valores de algún tipo de dato de Pascal.
  - Siempre retornan un valor de un tipo de Pascal.

¿Podré construir mis propias funciones?

PROCEDIMIENTOS EN PASCAL

- Ejemplos de algunos **procedimientos predefinidos** en Pascal:
  - **assign**(F, 'nombre');
  - **reset**(F);
  - **read**(A, B, C, D);
  - **readln;**
- Reflexionemos ahora sobre procedimientos:
  - Se los usa en una sentencia (no en expresiones).
  - Pueden recibir/retornar 0 o más valores.

¿Podré construir mis propios procedimientos?

RESOLUCION DE PROBLEMAS CON PRIMITIVAS

- Hemos visto cómo resolver **problemas simples**, escribiendo un **algoritmo** y luego un **programa** usando asignaciones, condiciones y repeticiones.
- En lo que resta de la materia veremos:
  1. Técnicas para resolver problemas más complejos.
  2. Cómo escribir algoritmos basados en primitivas y algoritmos que son primitivas.
  3. Cómo implementar en Pascal primitivas (funciones y procedimientos) y poder usar las dos técnicas anteriores.

Una **primitiva** es una operación o acción conocida, utilizada en un algoritmo o programa considerándola como básica.

FUNCIONES PREDEFINIDAS PARA REALES/ENTEROS EN PASCAL

Función	Descripción	Recibe	Retorna
abs	Valor absoluto	real o integer	El mismo tipo recibido
arctan	arctan en radianes	real o integer	real
cos	cosine de un radián	real o integer	real
exp	***e*** a una potencia	real o integer	real
ln	Algoritmo natural	real o integer	real
round	redondeo	real	integer
sin	Seno de un radián	real o integer	real
sqr	Cuandrado (square)	real o integer	El mismo tipo recibido
sqrt	square root (raíz cuad.)	real o integer	real
trunc	truncado	real o integer	integer

- http://wiki.freepascal.org/Standard_Functions

FUNCIONES PREDEFINIDAS PARA REALES/ENTEROS EN PASCAL

Función predefinida	Datos que recibe	Datos que retorna
abs	real o integer	mismo tipo recibido
round	real	integer
sqr	real o integer	mismo tipo recibido
sqrt	real o integer	real
trunc	real o integer	integer

- Estas funciones deben usarse en expresiones. Por ejemplo, puedo hacer:

```
if abs(num) < 100 then
   resultado := sqr(num);
```


FUNCION PARA LA POTENCIACION (RESTRINGIDA)

- Considere que queremos hacer una función "**potencia**" en Pascal, la cual permita calcular base ^{exponente} .
- Por ejemplo, para obtener en resultado el valor " 2^3 " poder hacer:

  resultado := potencia(2,3);

- El objetivo de este ejemplo es simplemente mostrar cómo se implementa una función en Pascal, por lo tanto, se implementará solamente para el caso en que base es un número entero y exponente un entero no negativo.

FUNCION PARA LA POTENCIACION (RESTRINGIDA)

- Para simplificar la explicación, nuestra nueva primitiva "potencia" asume que:
  - recibirá un número entero para la base,
  - recibirá un entero no negativo para el exponente, y
  - retornará un resultado entero.

Algoritmo: potencia
Datos que recibe: base (entero) y exponente (entero no neg.)
Dato que retorna: resultado (entero)
resultado := 1;
Repetir exponente veces:
    resultado := resultado * base.

FUNCION PARA LA POTENCIACION (RESTRINGIDA)

Algoritmo: potencia
Datos que recibe: base (entero) y exponente (entero no neg.)
Dato que retorna: resultado (entero)
resultado := 1;
Repetir exponente veces:
    resultado := resultado * base.

- Para implementar este algoritmo como una función en Pascal, hay una parte que ya hemos visto:

```
resultado := 1;
FOR aux:= 1 TO exponente DO
  resultado:= resultado * Base;
```

- Veremos ahora como indicar el resto de los elementos para obtener una función en Pascal.

FUNCIONES EN PASCAL

- A continuación se muestra una implementación para el algoritmo potencia.
- Vea la sintaxis de funciones en los **diagramas sintácticos**.

Algoritmo: potencia
Datos que recibe: base y exponente (entero)
Dato que retorna: resultado (entero)
resultado := 1;
Repetir exponente veces: resultado := resultado * base.

```
FUNCTION Potencia(Base,Exponente:integer):integer;
{Esta función retorna base elevado a la exponente}
VAR aux, resultado: integer; //variables propias de la func.
BEGIN
   resultado := 1;
   FOR aux:= 1 TO Exponente DO
     resultado := resultado * Base;
   Potencia:= resultado;
END;
```

CONCEPTOS

- En pascal una función tiene:
  1. Un **nombre** (con el cual se la invocará desde una expresión).
  2. **Parámetros** (entre los cuales estarán los datos de entrada).
  3. **Tipo del resultado** (que será el tipo de la función y determinará en que expresión podrá ser usada)
  4. **Variables locales** (que son propias de la función).
  5. Sentencias (también llamado "**cuerpo**" de la función).
  6. **Asignación** de una expresión al **nombre** de la función (al menos una vez). Es la forma de retornar un valor.

```
                1          2                3
FUNCTION Potencia(Base,Exponente:integer):integer;
{retorna base elevado a la exponente}
VAR aux, resultado: integer; 4//variables auxiliares
BEGIN
   resultado := 1;
   FOR aux:= 1 TO Exponente DO
5    resultado := resultado * Base;
   Potencia:= resultado;   6
END;
```

```
PROGRAM prueba; {prueba la función potencia}
VAR B,E, Pot :Integer;

FUNCTION Potencia(Base,Exponente:integer):integer;
{utilidad:calcula "base" elevado a la "exponente"}
VAR aux, resultado: integer;
BEGIN
   resultado := 1;
   FOR aux:= 1 TO Exponente DO
     resultado := resultado * Base;
   Potencia:= resultado;
END;

BEGIN
write('Ingrese base y exponente:');
repeat
  readln(B,E);
until E>=0;
Pot:=Potencia(B,E);
writeln(B,'a la',E,'=',pot);
writeln('2 a la ', E+1, '=', potencia(2,E+1));
END.
```

Variables globales
Tipo de la función
Parámetros formales
Documentación de la intención de esta función
Variables Locales
Asignación del resultado
Llamada a la función desde una expresión
Para hacer la traza vea la próxima explicación

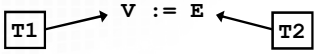
17

```
PROGRAM prueba_potencia;
VAR B,E, Pot :Integer;

FUNCTION Potencia(Base,Exponente:integer):integer;
{utilidad:calcula "base" elevado a la "exponente"}
VAR aux, resultado: integ
BEGIN
   resultado := 1;
   FOR aux:= 1 TO Exponente DO
     resultado := resultado * Base;
   Potencia:= resultado;
END;

BEGIN
write('Ingrese base y exponente:');
readln(B,E);
Pot:=Potencia(B,E);
writeln(B,'a la',E,'=',pot);
writeln('2 a la ', E+1, '=', potencia(2,E+1));
END.
```

Parámetros formales
Parámetros por valor: reciben una copia de los valores de los efectivos
Parámetros efectivos. Puedo utilizar cualquier constante, variable o expresión que sea asignación-compatible con el tipo del parámetro formal.
Para hacer la traza vea la próxima explicación

18

COMPATIBILIDAD DE ASIGNACIÓN



- Una expresión **E** de tipo **T2** es **asignación-compatible** con el identificador **V** de tipo **T1** si al menos una de las siguientes declaraciones es verdadera:
  1. **T1** y **T2** son idénticos.
  2. **T1** es real y **T2** es entero o subrango de entero.
  3. **T1** y **T2** son subrangos o enteros, y el valor de **E** es un valor permitido del tipo **T1**.

UNA FORMA DE HACER LA TRAZA DE PRUEBA_POTENCIA

Prueba_potencia	
B	?
E	?
pot	?

(1) Cuando comienza la ejecución del programa se crean las variables B, E, y pot.

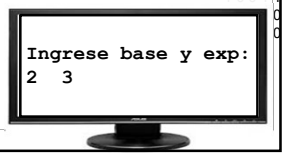
UNA FORMA DE HACER LA TRAZA DE PRUEBA_POTENCIA

Prueba_potencia	
B	?
E	?
pot	?

Llama a

potencia	

(1) Cuando comienza la ejecución del programa se crean las variables B, E, y pot. Consideremos el caso de prueba donde el usuario ingresa los valores 2 y 3.
Cuando la ejecución llega a la asignación **pot:=potencia(B,E);** se llama a al función potencia.



UNA FORMA DE HACER LA TRAZA DE PRUEBA_POTENCIA

Prueba_potencia	
B	2
E	3
pot	?

Llama a

potencia	
base	2
exponente	3
aux	?
resultado	?

(2) Inmediatamente después de invocar a la función **potencia** se crean los parámetros formales por valor (**base** y **exponente**) y las variables locales (**aux** y **resultado**) que son propias de la función.
El valor de **B** se copia a **base** y el valor de **E** se copia a **exponente**. De esta manera los dos parámetros formales tienen valor inicial antes de ejecutar la primera sentencia de la función.
Observe , sin embargo, que **aux** y **resultado** no tienen valor inicial antes de comenzar con la ejecución de las sentencias de la función.

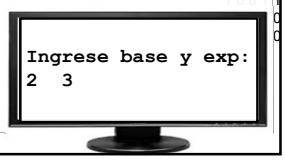
UNA FORMA DE HACER LA TRAZA DE PRUEBA_POTENCIA

Prueba_potencia	
B	2
E	3
pot	?

Llama a / Retorna 8

potencia	
base	2
exponente	3
aux	~~1,2~~,3
resultado	~~1,2,4~~,8

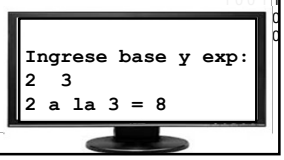
(3) Al ejecutarse la función potencia, la variable **resultado** inicia en 1. **Aux** es la variable de control de un FOR que se ejecuta 3 veces. Por lo tanto **resultado** toma los valores 2, 4 y 8.
La asignación **Potencia:= resultado;** tiene como efecto que al terminar la función retornará el valor de **resultado**, es decir 8.

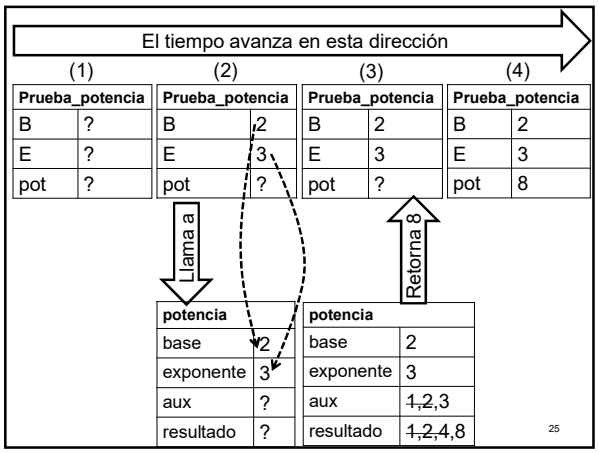


UNA FORMA DE HACER LA TRAZA DE PRUEBA_POTENCIA

Prueba_potencia	
B	2
E	3
pot	8

(4) Al finalizar la ejecución de la función **potencia**, todas las variables de la función desaparecen (la memoria es liberada para que la use cualquier programa en ejecución).
La ejecución continúa en el punto que había quedado en el programa prueba_potencia, esto es, en la asignación **pot:=potencia(B,E).**
El resultado de la función, un 8, es asignado entonces a la variable **pot**, y se muestran los valores por pantalla.





PROBLEMA PROPUESTO (LO MAS SIMPLE POSIBLE)

Problema propuesto: Escribir un programa que solicite al usuario tres números entre 1 y 200: N1, N2, y N3. El programa deberá mostrar por pantalla todas las combinaciones de un número elevado con otro. Esto es, N1 elevado a la N2, N1 a la N3, N2 a la N3, N2 a al N1, N3 a la N1 y N3 a la N2.

- Con la función potencia puedo resolver una parte.
- Se puede también hacer una primitiva especial para la carga de datos: una función que lea y valide que el dato de entrada está entre los topes "menor" y "mayor". Por ejemplo: FUNCTION leer_y_validar(menor, mayor: integer): integer;
- De manera tal que pueda hacer llamadas de este estilo:

  N1 := leer_y_validar(1, 200);

  N2 := leer_y_validar(1, 200);

FUNCIÓN PARA LEER Y VALIDAR

```
FUNCTION leer_y_validar(menor,mayor:integer):integer;
{lee usando readln un valor del buffer de entrada
hasta que el valor leído esté entre los topes
indicados como datos de entrada}
VAR
aux: integer; //para leer los valores a ser validados
BEGIN
 repeat
   write('Ingrese entero entre',menor,'y',mayor,':');
   readln(aux);
 until (aux>=menor) and (aux<=mayor);
 leer_y_validar:=aux; // retorno el valor leído y
                      validado
END;
```



```
PROGRAM problema2;
CONST tope_inf = 1; tope_sup = 200;
VAR n1,n2,n3: integer;
  FUNCTION Potencia(Base,Exponente:integer): integer;
  FUNCTION leer_y_validar(menor,mayor:integer):integer;
```

Hacer una traza

```
BEGIN
N1:=leer_y_validar(tope_inf, tope_sup);
N2:=leer_y_validar(tope_inf, tope_sup);
N3:=leer_y_validar(tope_inf, tope_sup);
write(N1,' a la ',N2,' es ',potencia(N1,N2)  );
write(N1,' a la ',N3,' es ',potencia(N1,N3)  );
write(N2,' a la ',N1,' es ',potencia(N2,N1)  );
write(N2,' a la ',N3,' es ',potencia(N2,N3)  );
write(N3,' a la ',N1,' es ',potencia(N3,N1)  );
write(N3,' a la ',N2,' es ',potencia(N3,N2)  );
END.
```

FUNCIONES EN PASCAL

- A continuación se muestra un algoritmo para identificar si una letra es mayúscula y una implementación con una función.

Algoritmo: esMayuscula
Datos que recibe: un carácter
Dato que retorna: verdadero/falso
si es una letra mayúscula retornar verdadero
de lo contrario retornar falso

```
FUNCTION EsMayuscula(letra: char): boolean;
{retorna verdadero si es letra mayúscula o false en caso contrario}
BEGIN
 IF ('A'<=letra) and (letra<='Z' ) or
    (letra = chr(165)) THEN // Ñ
   EsMayuscula:=true
 ELSE
   EsMayuscula:=false
END;
```

INVOCACION A FUNCIONES (EN EXPRESIONES)

```
FUNCTION EsMayuscula(letra: char): boolean;
{retorna verdadero si es letra mayúscula o false en caso contrario}
BEGIN
 IF ('A'<=letra) and (letra<='Z' ) or
    (letra = chr(165)) THEN // Ñ
   EsMayuscula:=true
 ELSE
   EsMayuscula:=false
END;
```

Algunos ejemplos de invocación (siempre en expresiones):

```
VAR ch: char;
    es_mayu:boolean;
  …
  read(ch);
  es_mayu := EsMayuscula(ch);
  writeln(EsMayuscula(ch));
  IF (EsMayuscula(ch) or (ch='@')) THEN …
  WHILE not EsMayuscula(ch) and not EOF(…) DO …
```

CONCEPTOS: INVOCACION A FUNCIONES

- Para **llamar** (invocar o usar) a una función se debe:
  1. Utilizar la llamada en una **expresión**.
  2. Coincidir la **cantidad** de **parámetros**; y el **tipo de dato** de cada uno de los parámetros efectivos debe ser asignación compatible con el parámetro formal correspondiente.
  3. El **tipo** del **resultado** debe ser **compatible** con el tipo de la **expresión** en la que se lo **llama**.
- Por ejemplo, considerando:

```
FUNCTION Potencia(Base,Exponente:integer):integer;
```

todas estas llamadas son correctas:

OK

```
aux:=Potencia(2,7);
write(Potencia(2,7));
write(potencia(1+1, 14 div 2));
if potencia(2,7) > tope then…
```

31

CONCEPTOS: INVOCACION A FUNCIONES

- Dado la siguiente función:

```
FUNCTION Potencia(Base,Exponente:integer):integer;
```

todas estas llamadas tienen **errores de programación**.

```
Potencia(2,7);
write(potencia(2.1,3)
write(Potencia(2));
write(potencia(2,3,1));
write(potencia('x',2));
if potencia(2,7) then…
```

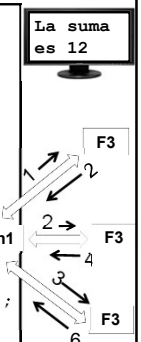
MAL

- Error: no está en una expresión
- Error: el primer parámetro es de tipo real
- Error: falta un parámetro
- Error: sobra un parámetro
- Error: parámetro no compatible
- Error: tipo del resultado no compatible

32

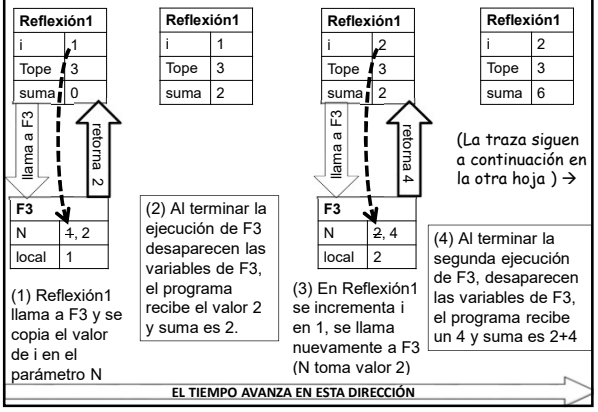
Para reflexionar sobre el pasaje de parámetros, las variables locales y globales, con respecto a la **dinámica** en la ejecución realice la traza de los siguientes programas:

```
program Reflexion1;
{Para reflexionar sobre la dinámica con
una traza}
var i, tope, suma:integer;
 Function F3(N:integer):integer;
 var {F3 modifica local y N}
  local:integer;
 begin
  local:= N; N:= N + local; F3:= N;
 end;
begin
tope:= 3;
Suma:= 0; {F3 es llamada 3 veces}
for i := 1 to tope do suma:= suma + F3(i);
Writeln('La suma es ', suma);
end.
```

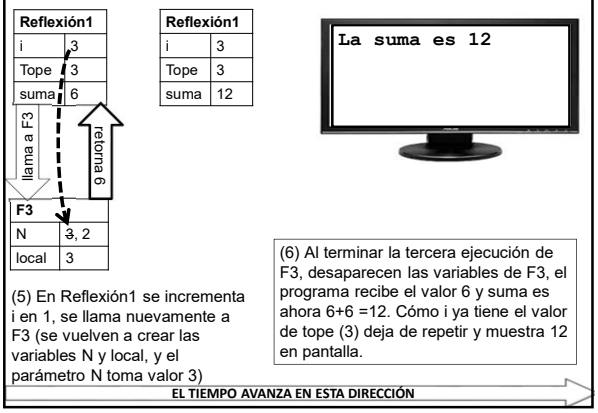


33

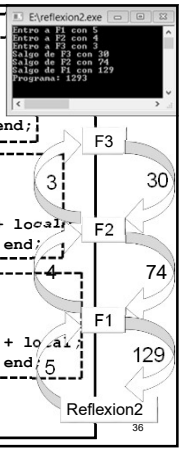
Una forma de hacer la traza de reflexión1



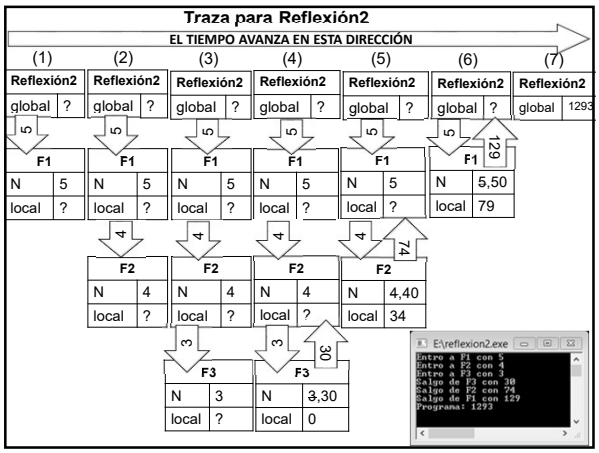
(continuación de la traza)




```
program reflexion2; var global:integer;
 Function F3(N:integer):integer;
 var local:integer;
 begin writeln('Entro a F3 con', N);
 local:= 0;  N:= N * 10; F3:= N + local;
 writeln('Salgo de F3 con', N + local); end;

 Function F2(N:integer):integer;
 var local:integer;
 begin writeln('Entro a F2 con', N);
 local:= N+ F3(N-1); N:=N * 10;  F2:= N + local;
 writeln('Salgo de F2 con ', N + local); end;

 Function F1(N:integer):integer;
 var local:integer;
 begin writeln('Entro a F1 con', N);
 local:=N+F2(N - 1); N:= N * 10;  F1:= N + local;
 writeln('Salgo de F1 con ', N + local); end;

begin global := 3 + F1(5) * 10;
write('Programa:', global); end.
```



36


Traza para Reflexión2
EL TIEMPO AVANZA EN ESTA DIRECCIÓN
(1)
(2)
(3)
(4)
(5)
(6)
(7)
Reflexión2
global ?
global 1293
5
129
F1
N 5
local ?
N 5,50
local 79
4
74
F2
N 4
local ?
N 4,40
local 34
3
30
F3
N 3
local ?
N 3,30
local 0
E:\reflexion2.exe
Entro a F1 con 5
Entro a F2 con 4
Entro a F3 con 3
Salgo de F3 con 30
Salgo de F2 con 74
Salgo de F1 con 129
Programa: 1293