Mary Allen Wilkes (1937)

- Creadora (1965) del primer ordenador personal para trabajo en casa
- Desarrolladora de sistemas operativos e linguaxe ensamblador (LAP6)
- Traballou no MIT e na Univ. Washington

Tipos de datos definidos por usuario

- Podemos definir tipos de datos agregados.

```
type nome
    tipo1 :: var1
    tipoN :: varN
end type nome
```

- Defínense en módulos para poder usarse en varios subprogramas.

```
module modulo_persoa
type persoa
      character(10) :: nome
      integer :: idade
end type persoa
end module modulo_persoa
```

```
program exemplo_tipos
use modulo_persoa
type(persoa) :: p=persoa('carlos',14)
call mostra(p)
end program exemplo_tipos

subroutine mostra(p)
use modulo_persoa
type(persoa),intent(in) :: p
print *,'nome=',p%nome,'idade=',p%idade
end subroutine mostra
```

- Declaración: *type(nome) :: x=nome(y,z)*
- Acceso a campos: *x%y,x%z*

Sobrecarga de operadores

- Define un operador a medida para un dato agregado.
- Bloque *interface operator*.
- Función que define o operador aritmético ou relacional.

```
program principal
use exemplo
type(persoa) :: x=persoa('alba',65.2,1.84)
type(persoa) :: y=persoa('clara',70.1,1.98)
if(x>y) then
      print *,x%nome,'>',y%nome
else
      print *,x%nome,'<=',y%nome
end if
stop
end program principal
```

```
module exemplo
      type persoa
            character(10) :: nome
            real :: peso,altura
      end type persoa
      interface operator (>)
            module procedure maior
      end interface
contains
      logical function maior(x,y)
      type(persoa),intent(in) :: x,y
      if(x%peso/x%altura > y%peso/y%altura) then
            maior=.true.
      else
            maior=.false.
      end if
      end function maior
end module
```

Representación gráfica (I): curva 2D (función de 1 variábel)

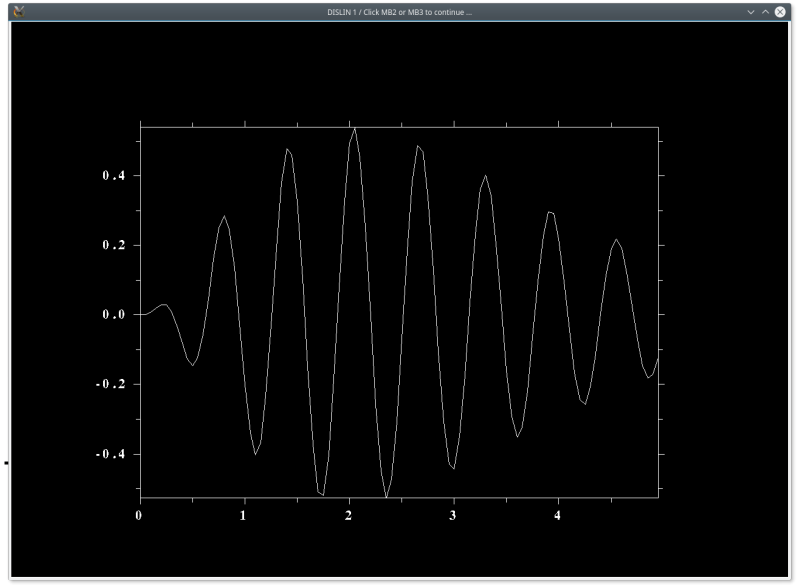
- Libraría Dislin: https://www.dislin.de
- Tamén o podes descargar dende: http://ftp5.gwdg.de/pub/grafik/dislin/linux/i586_64/

Helmut Michels: **The data plotting software DISLIN : Version 11,** ISBN 9783868585179, Sinatura 1203 232 (Bibl. Fac. Matemáticas)

- Instala paquete *libmotif-dev*
- Descargar o arquivo .deb dende https://www.dislin.de/i586_64.html e executa como root: *dpkg -i arquivo.deb*
- *export LD_LIBRARY_PATH=/usr/local/dislin* (directorio donde está o arquivo *libdislin.so*)
- Compilación: *f95 -I/usr/local/dislin/gf curva2d.f90 -ldislin*
- Curva 2D: $y=f(x)=t^2e^{-t}sen\ 10t$: vectores x, y con coordenadas de puntos

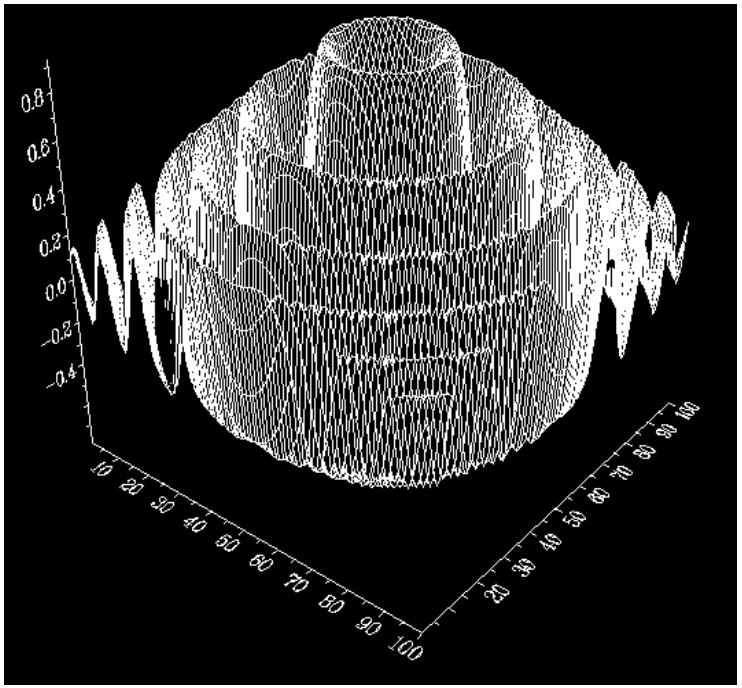
Programación estruturada en Fortran

```
program curva2d
use dislin
integer,parameter :: n=100
real,dimension(n) :: x,y
a=0;b=5;h=(b-a)/n;t=a
do i=1,n
      x(i)=t;y(i)=t**2*exp(-t)*sin(10*t)
     t=t+h
end do
call metafl('xwin')
call disini()
call qplot(x,y,n)
stop
end program curva2d
```



Representación gráfica (II): superficie 3D (función 2 variábeis)

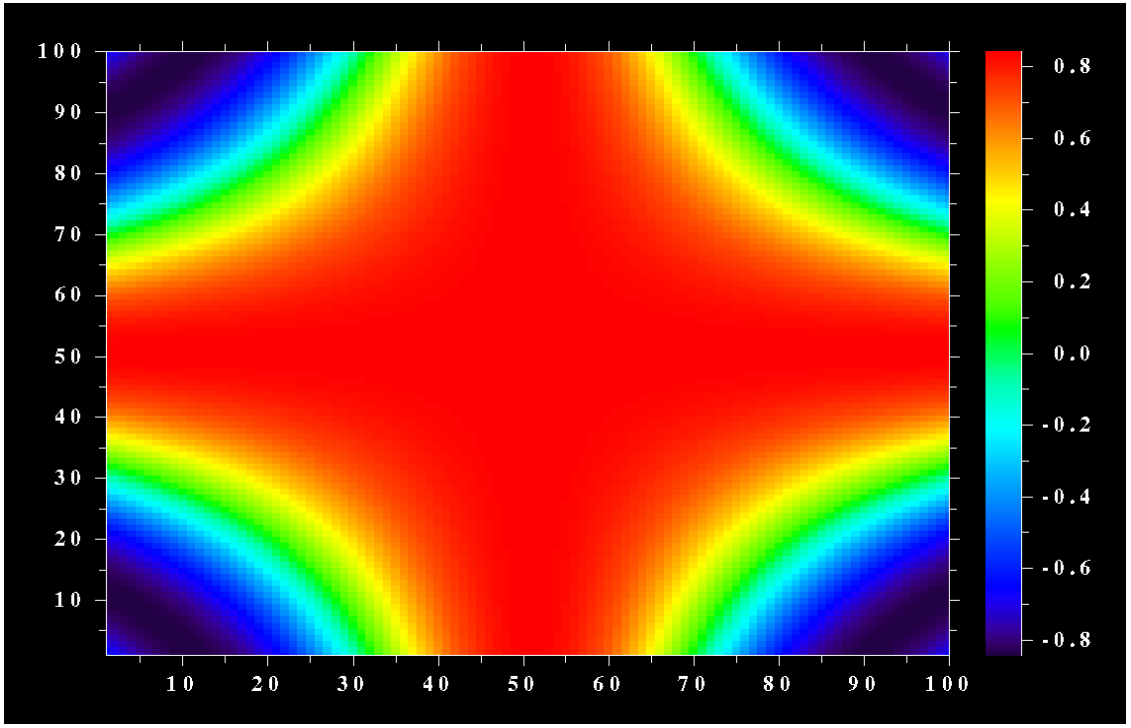
- Superficie 3D: matriz *z* con valores de función *z=f(x,y)*, *n* valores en cada dimensión.
- Exemplo: $z=f(x,y)=sen(5(x^2+y^2))exp(-(x^2+y^2)/4)$.




```
program superficie3d
use dislin
integer,parameter :: n=100
real,dimension(n) :: x,y
real,dimension(n,n) :: z
a=-2;b=2;h=(b-a)/n;t=a
do i=1,n
      x(i)=t;y(i)=t;t=t+h
end do
do i=1,n
      do j=1,n
            tx=x(i);ty=y(j);
          z(i,j)=sin(5*(tx**2+ty**2))*
             exp(-(tx**2+ty**2)/4)
      end do
end do
call metafl('xwin')
call disini()
call qplsur(z,n,n)
stop
end program superficie3d
```

Representación gráfica (III): mapa de calor (función 2 variábeis)

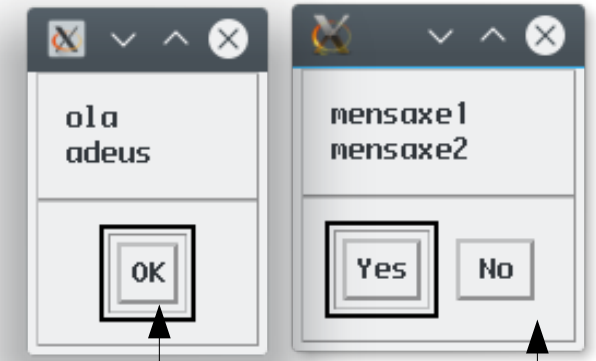
- Mapa de calor: diagrama de cores que representan o valor dunha función $z=f(x,y)$, azul=valores baixos, vermello=valores altos.
- Exemplo: $z=f(x,y)=sen(5(x^2+y^2))exp(-(x^2+y^2)/4)$.




```
program mapa_calor
use dislin
integer,parameter :: n=100
real,dimension(n) :: x,y
real,dimension(n,n) :: z
a=-2;b=2;h=(b-a)/n;t=a
do i=1,n
      x(i)=t;y(i)=t;t=t+h
end do
do i=1,n
      do j=1,n
          tx=x(i);ty=y(j);
          z(i,j)=sin(cos(tx*ty))
      end do
end do
call metafl('xwin')
call disini()
call qplclr(z,n,n)
stop
end program mapa_calor
```

Interface gráfica de usuario: ventá emerxente con mensaxe e botón

export LD_LIBRARY_PATH=/usr/local/dislin
Compilación: *f95 -I/usr/local/dislin/gf curva2d.f90 -ldislin*



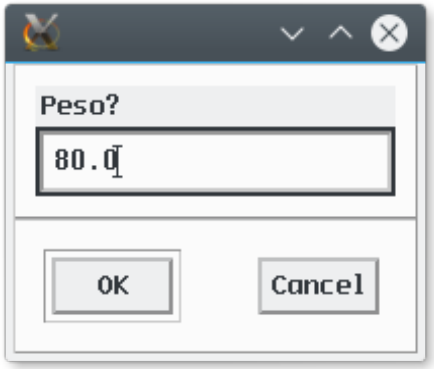
- Compoñentes gráficas (*widgets*).
- Comezando: ventás emerxentes.
- Mensaxe de varias liñas (separadas con |) e un botón OK.
- Mensaxe con dous botóns (Yes e No), podes comprobar que botón se pulsou.

```
program mensaxe
use dislin
call dwgmsg('ola|adeus')
stop
end program mensaxe
```

```
program boton
use dislin
call dwgbut('mensaxe1|mensaxe2', ival)
if(ival==0) then
      print *,'pulsaches non'
else
      print *,'pulsaches si'
end if
stop
end program boton
```

Ventá emerxente con entrada de texto/números

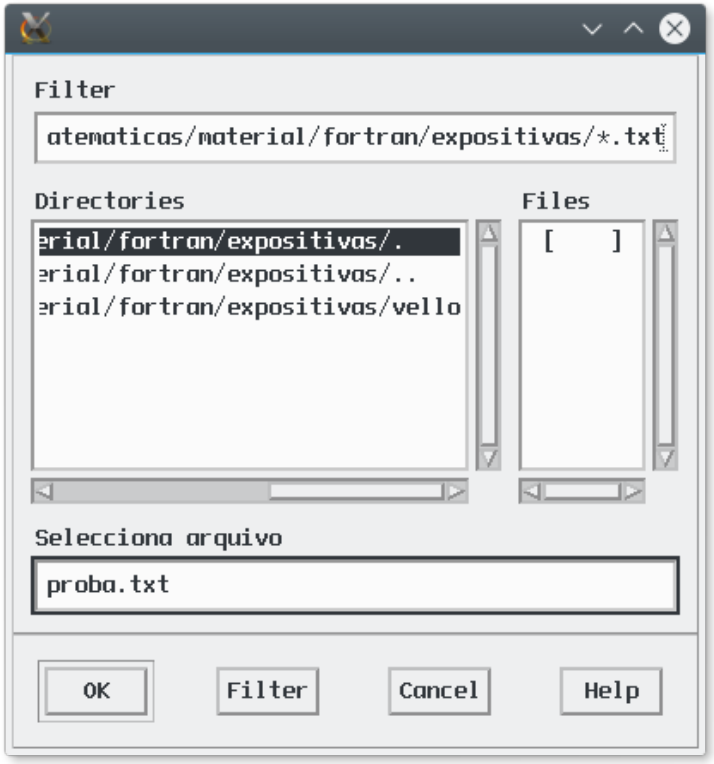
- Entrada de texto/números.
- A entrada almacénase como cadea de caracteres.
- Logo podes converter a número (enteiro, real).
- Botóns *OK* e *Cancel*: o programa pode comprobar se se cancelou ou se introduciu un dato.




```
program entrada_texto
use dislin
character(100) :: s='80.0'
integer :: estado
call dwgtxt('Peso?',s)
call dwgerr(estado)
if(estado==0) then
      read (s,*) p
      print *,'introduciches ',p
else
      print *,'cancelaches'
end if
stop
end program entrada_texto
```

Ventá emerxente con selector de arquivos

- Permite seleccionar un arquivo no directorio actual e navegar polos directorios
- Indica un tipo de arquivo e nome por defecto.
- Permite cancelar.




```
program selector_arquivo
use dislin
character(100) :: f='proba.txt'
integer :: estado
call dwgfil('Selecciona arquivo',f,'*.txt')
call dwgerr(estado)
if(estado==0) then
      print *,'seleccionaches ',f
else
      print *,'cancelaches'
end if
stop
end program selector_arquivo
```

Ventá emerxente con lista de opcións

- Mostra unha lista e permite seleccionar un dos seus elementos, indicando o elemento seleccionado por defecto.
- Permite cancelar.


```
program lista_opcions
use dislin
integer :: sel=2,estado
call dwglis('lista','Carlos|Luis|Alberto',sel)
call dwgerr(estado)
if(estado==0) then
      print *,'seleccionaches ',sel
else
      print *,'cancelaches'
end if
stop
end program lista_opcions
```

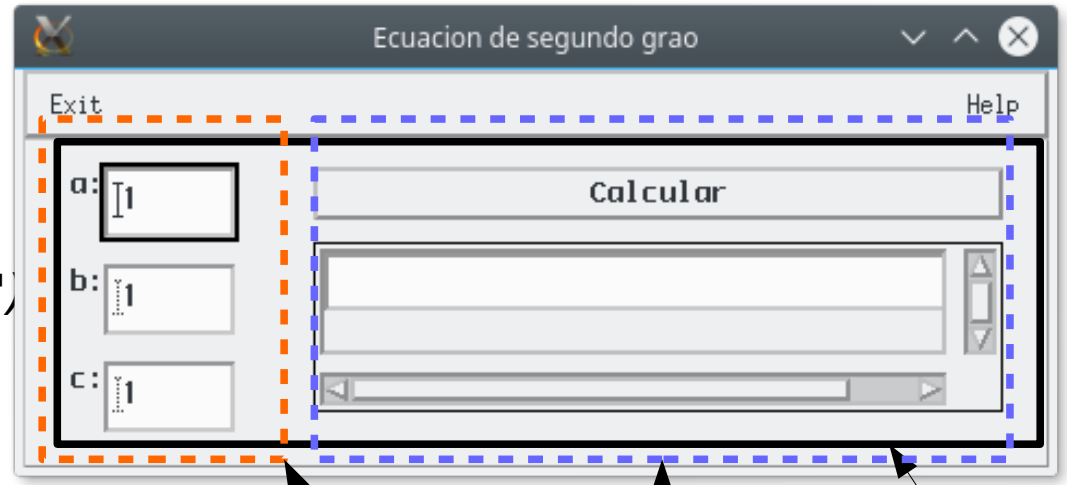
Interfaz de usuario con deseño complexo

- Subrutina *wgini(tipo,id)*: inicializa as compoñentes gráficas e crea un contedor de compoñentes, tipo=’vert’ ou ’hori’, segundo o aliñamento das compoñentes.

call wgini('vert',v)
call wgltxt(v, 'x: ', '1', 50,t1) ! Cadro de texto nº 1
call wgltxt(v,'x^0.3+x-1: ',s,50,t2) ! Cadro de texto nº 2 (debaixo do actual)

- Se uso ‘*hori*’ (‘*vert*’) en *wgini*, as compoñentes gráficas engádense de esquerda a dereita (de arriba a abaixo).
- Cada contedor ou compoñente ten un *id* (enteiro): *v*,*t1*,*t2*.
- Podes poñer un contedor horizontal dentro doutro vertical, e viceversa.
- Remata as compoñentes gráficas con *wgfin*.

Deseño da interfaz: contedores horizontais e verticais con compoñentes




```
program ec2grao
use dislin
integer :: ip, ipv1, ipv2, iph
call swgtit ('Ecuacion de segundo grao')
call wgini('vert', ip) ! inicializa GUI
call wgbas(ip, 'hori', iph) ! horizontal
call swgwth(10) ! pon anchura
call wgbas(iph, 'vert', ipv1) ! vertical
call wgltxt(ipv1, 'a:', '1', 80, id_a) ! pon widget para ler texto
call wgltxt(ipv1, 'b:', '1', 80, id_b)
call wgltxt(ipv1, 'c:', '1', 80, id_c)
call swgwth(40) ! pon anchura
call wgbas(iph, 'vert', ipv2) ! contedor vertical
call wgpbut(ipv2,'Calcular', id_button) ! botón no GUI
call wgstxt(ipv2, 2, 1, id_txt) ! cadro para mostrar texto
call wgfin() ! mostra GUI
stop
end program ec2grao
```

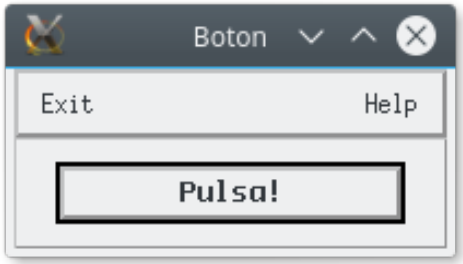
Contedor vertical *ipv2*

Contedor horizontal *ip*

Contedor vertical *ipv1*

Execución de accións asociadas a compoñentes

- Cadros de texto, botóns: ao pulsar *Intro* (p.ex. cadro de texto) ou co rato (p.ex. botón), executar accións.
- Subprogramas retro-chamados (*callback*). Asocia subprograma con compoñente gráfica.
- *call swgcbk(id,subprograma)*: asocia o subprograma á pulsación da compoñente gráfica *id*.
- O subprograma debe ser declarado *external*.




```
subroutine pulsa(b)
integer,intent(in) :: b
print *, 'has pulsado!'
return
end subroutine pulsa
```

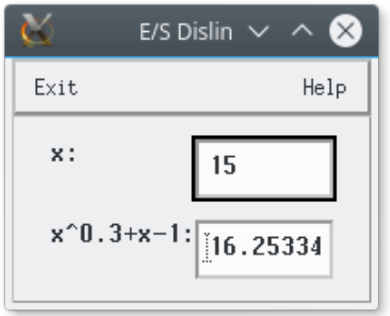
```
program widget_boton
use dislin
integer :: v,b
external pulsa
call swgtit ('Boton')
call wgini('vert',v)
call wgpbut(v,'Pulsa!',b)
call swgcbk(b,pulsa)
call wgfin
stop
end program widget_boton
```

Exemplo: entrada e saída gráfica

- Libraría **Dislin** (non incluída en **Gfortran**).
- Cadros de texto para ler *x* e para mostrar $x^{0.3}+x-1$. Escribe o número *x* e pulsa *Intro*.

```
program entrada_saida_grafica
use dislin
character(100) :: s
integer :: v,t1,t2
external calcula
common t1,t2
call swgtit('E/S Dislin')
call wgini('vert',v)
call wgltxt(v, 'x: ', '1', 50,t1)
call swgcbk(t1,calcula)
write (s,*) x**0.3+x-1
call wgltxt(v,'x^0.3+x-1: ',s,50,t2)
call wgfin
stop
end program entrada_saida_grafica
```

```
subroutine calcula(id)
integer,intent(in) :: id
common t1,t2
character(len=50) :: s
call gwgflt(t1,x)
y=x**0.3+x-1
call swgflt(t2,y,5)
return
end subroutine calcula
```



Le no 1º cadro

Escribe no 2º cadro

Cadro de texto para ler x

Executase cando se pulsa Intro no 1º cadro de texto

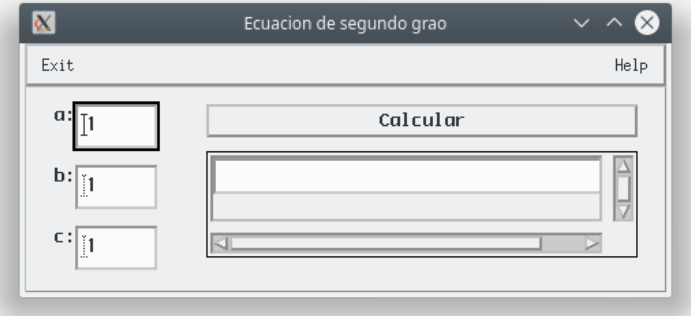
Cadro de texto para escribir $x^{0.3}+x-1$

Interfaz de usuario para o programa da ecuación de 2º grao

```
module comun ! para definir variables globais o programa
integer :: id_a, id_b, id_c, id_txt
end module comun
```

```
program ec2grao
use dislin
use comun
integer :: ip, ipv1, ipv2, iph, boton
external :: calcula
call swgtit('Ecuacion de segundo grao')
call wgini('vert', ip) ! inicializa GUI
call wgbas(ip, 'hori', iph) ! horizontal
call swgwth(10) ! pon anchura
call wgbas(iph, 'vert', ipv1) ! vertical
call wgltxt(ipv1, 'a:', '1', 80, id_a) ! pon widget para ler texto
call wgltxt(ipv1, 'b:', '1', 80, id_b)
call wgltxt(ipv1, 'c:', '1', 80, id_c)
call swgwth(40) ! pon anchura
call wgbas(iph, 'vert', ipv2) ! contedor vertical
call wgpbut(ipv2,'Calcular', boton) ! botón no GUI
call swgcbk(boton, calcula) ! conecta botón-subprograma
call wgstxt(ipv2, 2, 1, id_txt) ! cadro para mostrar texto
call wgfin ! mostra GUI
stop
end program ec2grao
```

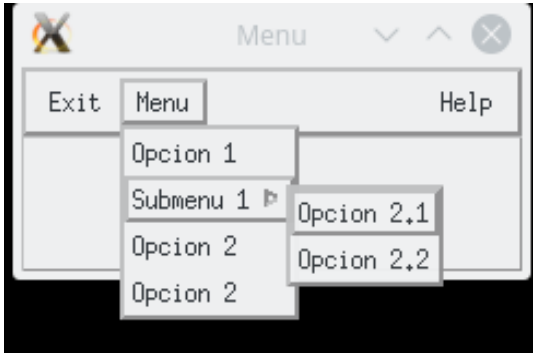
```
subroutine calcula(id)
use comun
integer, intent(in) :: id
character(50) :: s, ecuacion
! lee o número da etiqueta de texto
call gwgflt(id_a, a)
call gwgflt(id_b, b)
call gwgflt(id_c, c)
! calcula a solución (carácter)
s=ecuacion(a,b,c)
! mostra a solución no cadro de texto
call swgtxt(id_txt,s)
return
end subroutine calcula
```



Compoñente gráfica menú

- Menú con ítems e submenús:
- *call wgpop(id1,'nome',id)*: crea menú ou submenú no contedor *id1*
- *call wgapp(m,'nome',id)*: engade elemento *id* ao menú ou submenú *m*.
- Cada elemento debe ter un subprograma que se execute cando é seleccionado.

```
program widget_menu
use dislin
integer :: v,m,o1,sm1,o2,o21,o22,o3
call swgtit ('Menu')
call wgini('vert',v)
call wgpop(v,'Menu',m)
call wgapp(m,'Opcion 1',o1)
call wgpop(m,'Submenu 1',sm1)
call wgapp(sm1,'Opcion 2.1',o21)
call wgapp(sm1,'Opcion 2.2',o22)
call wgapp(m,'Opcion 2',o2)
call wgapp(m,'Opcion 2',o3)
call wgfin
stop
end program widget_menu
```

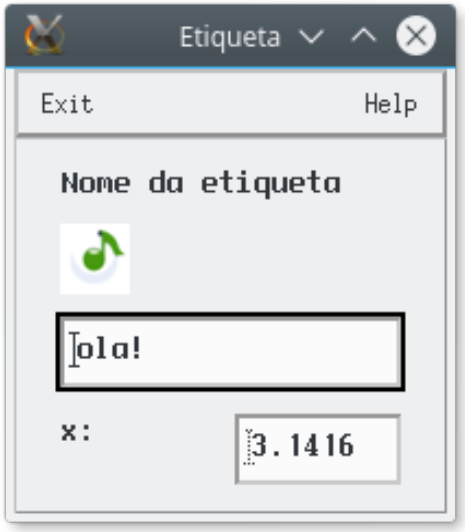


Etiqueta e cadro de texto

- Etiqueta: *call wglab(id1,'nome',id)*. Tamén con icono no canto de texto: *wgicon*
- Cadro de texto: *call wgtxt(id1,'texto',id)*
- Tamén con etiqueta: *wgltxt*

```
program widget_varios
use dislin
integer :: v,l,i,t1,t2
external le_t1
call swgtit ('Etiqueta e cadro de texto')
call wgini('vert',v)
call wglab(v,'Nome da etiqueta',l)
call wgicon(v,'Nome do icono',0,0,'icono.ico',i)
call wgtxt(v,'ola!',t1)
call swgcbk(t1,le_t1)
call wgltxt(v,'x: ','3.1416',50,t2)
call wgfin
stop
end program widget_varios
```

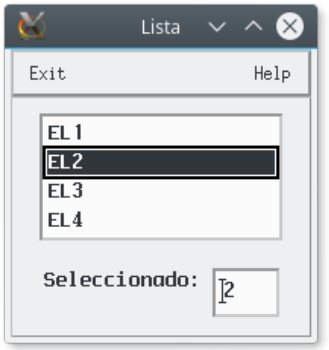
```
subroutine le_t1(t1)
integer,intent(in) :: t1
character(100) :: s
call gwgtxt(t1,s)
print *, 'escribiches ',s
return
end subroutine le_t1
```



Le o cadro de texto ao pulsar Intro e móstrao na terminal

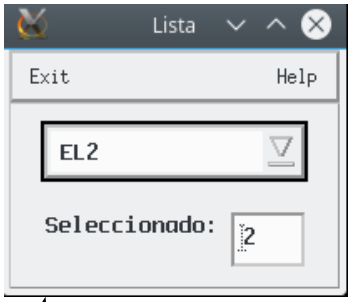
Lista e lista despregábel

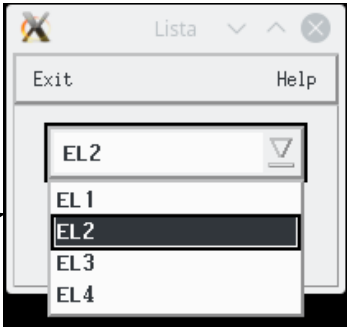
- *call wglis(id1,'E1|E2|E3|E4',sel,id)*: crea lista en *id1*.
- *call swgcbk(id,accion)*: acción ao seleccionar un elemento da lista.
- *call gwglis(id,i)*: i=elemento seleccionado.




```
program widget_lista
use dislin
integer :: v,l,sel=3,t
character(30) :: s
common t
external le_lista
call swgtit ('Lista')
call wgini('vert',v)
call wglis(v,'EL1|EL2|EL3|EL4',sel,l)
write (s,'(i0)') sel
call wgltxt(v,'Seleccionado: ',s,30,t)
call swgcbk(l,le_lista)
call wgfin
stop
end program widget_lista
```

```
subroutine le_lista(l)
integer,intent(in) :: l
common t
integer :: sel
character(20) :: s
call gwglis(l,sel)
write (s,'(i0)') sel
call swgtxt(t,s)
return
end subroutine le_lista
```

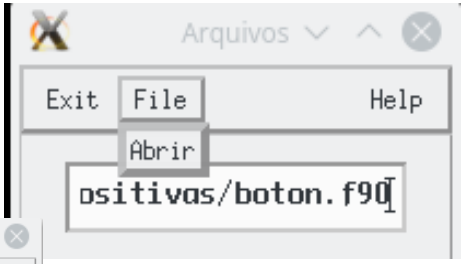


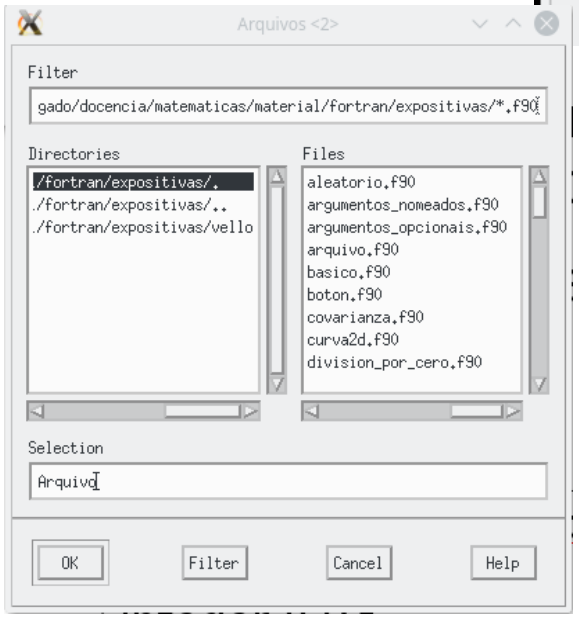


- Lista despregábel: *wdglis(...)*

Menú con selector de arquivos

- *call wgfil(id1,'Abrir','Arquivo','*.f90',id)*: crea un menú *File* cun elemento *Abrir*.
- Cando se pulsa, abre un selector de arquivos, mostrando so os arquivos coa extensión indicada (neste caso, **.f90*).
- Seleccionado o arquivo, aparece no cadro de texto

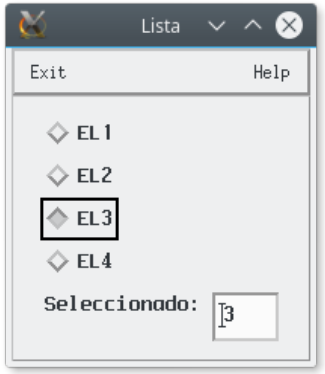





```
program widget_selec_arquivo
use dislin
integer :: v,f
call swgtit ('Arquivos')
call wgini('vert',v)
call wgfil(v,'Abrir','Arquivo','*.f90',f)
call wgfin
stop
end program widget_selec_arquivo
```

Lista de elementos excluintes

- *call wgbox(id1,'E1|E2|E3|E4', sel,id)*: crea lista con elementos que son botóns activábeis excluíntes (se activas un, desactívanse os demáis).
- *call swgcbk(id,accion)*: acción ao seleccionar un elemento da lista.
- *call gwgbox(id,i)*: i=elemento seleccionado.




```
subroutine le_lista(l)
integer,intent(in) :: l
common t
integer :: sel
character(20) :: s
call gwgbox(l,sel)
write (s,'(i0)') sel
call swgtxt(t,s)
return
end subroutine le_lista
```

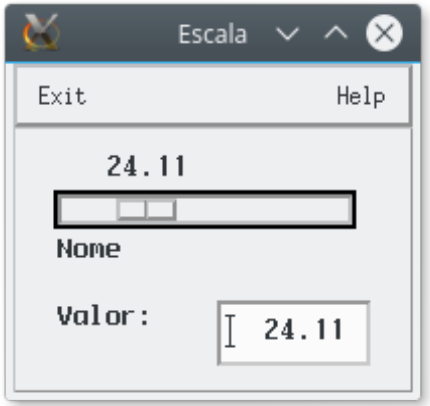
```
program widget_wgbox
use dislin
integer :: v,l,sel=3,t
character(30) :: s
common t
external le_lista
call swgtit ('Lista')
call wgini('vert',v)
call wgbox(v,'EL1|EL2|EL3|EL4',sel,l)
write (s,'(i0)') sel
call wgltxt(v,'Seleccionado: ',s,30,t)
call swgcbk(l,le_lista)
call wgfin
stop
end program widget_wgbox
```

Escala

- Regra horizontal na que podes seleccionar un valor.
- call wgscl(id1,’nome’,vmin,vmax,vdef,ndix,id): valores mínimo, máximo e por defecto (reais), nº díxitos.

```
program widget_scale
use dislin
integer :: v,sc,t
character(10) :: s
common t
external le_scale
call swgtit ('Escala')
call wgini('vert',v)
call wgscl(v,'Nome',0.,100.,75.0,2,sc)
call gwgscl(sc,sel)
write (s,'(f7.2)') sel
call wgltxt(v,'Valor: ',s,50,t)
call swgcbk(sc,le_scale)
call wgfin
stop
end program widget_scale
```

```
subroutine le_scale(l)
integer,intent(in) :: l
common t
character(20) :: s
call gwgscl(l,sel)
write (s,'(f7.2)') sel
call swgtxt(t,s)
return
end subroutine le_scale
```

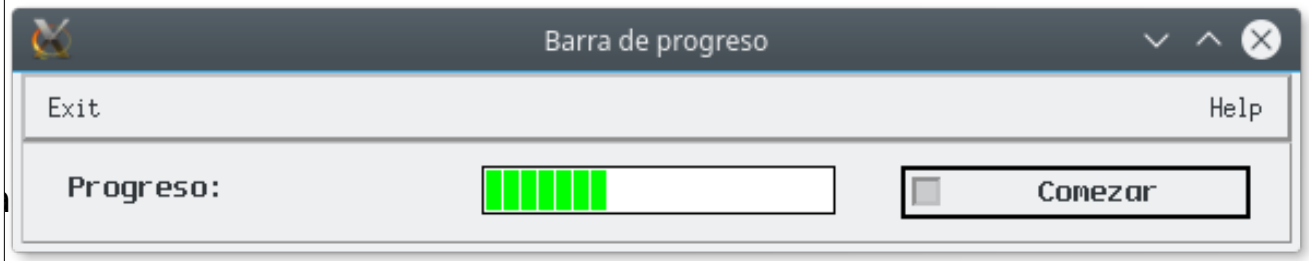


Barra de progreso

- Regra horizontal que indica o progreso dun cálculo (0-100%)
- Moi útil para cálculos numéricos lentos.
- *call wgpbar(id1,ini,fin,paso,id)*
- Retro-subprograma, asociado a un botón de inicio, que actualiza o progreso da barra.

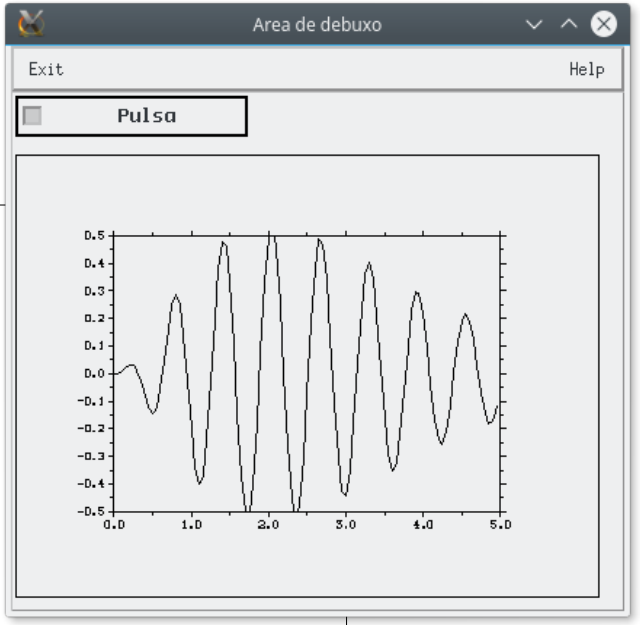
```
program widget_progress_bar
use dislin
integer :: h,l,pb,b,n=20
common pb,n,b,l2
external actualiza_pb
call swgtit ('Barra de progreso')
call wgini('hori',h)
call wglab(h,'Progreso:',l)
call wgpbar(h,0.,100.,100./n,pb)
call wgbut(h,'Comezar',0,b)
call swgcbk(b,actualiza_pb)
call wgfin
stop
end program widget_progress_bar
```

```
subroutine actualiza_pb(id)
integer,intent(in) :: id
common pb,n,l2
do i=1,n
      call sleep(1)
      call swgval(pb,100.*i/n)
end do
call swgbut(id,0)
return
end subroutine actualiza_pb
```



Área de debuxo (*draw*)

- *call wgdraw(id1,id)*: a retro-subrutina do botón representa unha función na compoñente gráfica.
- Fixa o seu tamano.




```
program widget_draw
use dislin
integer :: v,b,dr
common dr
external representa
call swgtit ('Area de debuxo')
call wgini('form',v)
call wgbut(v,'Pulsa',0,b)
call swgcbk(b,representa)
call swgpos(0,40)
call swgsiz(400,300)
call wgdraw(v,dr)
call wgfin
stop
end program widget_draw
```

```
subroutine representa(id)
integer,intent(in) :: id
integer,parameter :: n=100
real :: x(n),y(n)
common dr
call metafl('cons')
call setxid(dr,'widget')
call scrmod('reverse')
call disini
a=0;b=5;h=(b-a)/n;t=a
call graf(a,b,a,1.,-0.5,0.5,-0.5,0.1)
do i=1,n
   x(i)=t;y(i)=t**2*exp(-t)*sin(10*t);t=t+h
end do
call curve(x,y,n)
call disfin
return
end subroutine representa
```

Creación dunha libraría en Fortran

- Libraría: código máquina de moitos subprogramas (en Fortran) empaquetado nun arquivo
- Non contén o código fonte (non se pode depurar ou ver como opera).
- Proporciónanos subprogramas que permiten facer operacións (p.ex., determinantes, resolución de sistemas de ecuacións, ...)
- Para usar unha libraría, hai que enlazar (*link*) o noso programa coa libraría
- O compilador *f95* xa usa algunhas librarías incluídas co compilador (p.ex. funcións intrínsecas)

Librarías estáticas

- Ficheiro con extensión *.a*: *libproba.a*. Usado só na compilación:

  f95 -L. programa.f90 -lproba

- O código máquina da libraría empótrase no programa executábel.
- Non se necesita nada para a execución: *a.out*
- Programa executábel de tamano grande (carga en memoria RAM máis lenta).
- Listado de arquivos *.o* contidos en libraría *.a*: *ar tv libproba.a (ou nm X.a*, ou *readelf -s X.a)*

Librarías dinámicas

- Ficheiro con extensión *.so*: *libproba.so*
- O código máquina da libraría úsase na compilación e na execución:

  f95 -L. programa.f90 -lproba

  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.

  a.out

- Executábel máis pequeno (carga máis rápida en memoria).
- Listado de arquivos *.o*: *nm X.so* ou *readelf -s X.so*.
- Non se pode executar sen que a variábel de entorno indique a ruta onde se atopan o(s) arquivo(s) **.so**
- Ver a variábel: *echo $LD_LIBRARY_PATH*

Creación dunha libraría estática en Fortran (I)

- Edición/compilación/depuración de código
- Compilación sen enlazado dos arquivos que conteñen subprogramas (non incluir o programa principal). Crea só arquivos obxecto (.o), non crea executábel:

  f95 -c arquivo.f90 (crea *arquivo.o*)

- Empaquetado de arquivos *.o* e creación de libraría (***libXX.a***, debe comezar por *lib*)

  *ar qv libXX.a *.o*

- Para ver arquivos .o empaquetados: *ar tv libXX.a*

Creación dunha libraría estática en Fortran (II)

- É recomendábel meter unha función ou subrutina en cada *arquivo .o*
  - Alomenos os subprogramas que van ser usados dende fóra da libraría).
- Así coinciden os nomes dos arquivos *.o* empaquetados en *libXX.a* cos nomes dos subprogramas que proporciona a libraría.
- Así podemos saber, co comando *ar tv libXX.a*, os subprogramas que contén esa libraría.

Exemplo de creación e uso dunha libraría estática

- Libraría *libestatistica.a* que proporcione subprogramas para calcula-la media, desviación, mediana e ordear un vector
- Arquivos media.f90, desviacion.f90, mediana.f90, ordea.f90, principal.f90 (descárgaos)
- Comando de compilación (sen enlazado):

  f95 -c media.f90 desviacion.f90 mediana.f90 ordea.f90

- Empaquetado da libraría: *ar qv libestatistica.a *.o*
- Listado de arquivos *.o*: *ar tv libestatistica.a* (ou *readelf -s X.a*)
- Para enlazar o programa *principal.f90* coa libraría:

  f95 -L. principal.f90 -lestatistica

- Execución: *a.out*

Exemplo de creación e uso dunha libraría dinámica

- Libraría *libestatistica.so*. Mesmos arquivos *.f90* de antes.
- Comando de compilación (sen enlazado):

  f95 -fpic -c media.f90 desviacion.f90 mediana.f90 ordea.f90
- Empaquetado: *f95 -shared -o libestatistica.so *.o*
- Listado de arquivos *.o*: *nm libestatistica.so* ou *readelf -s X.so*
- Uso da libraría dende programa *principal.f90*:

  f95 -L. principal.f90 -lestatistica
- Engade a ruta de libraría ao *LD_LIBRARY_PATH*:

  export LD_LIBRARY_PATH = $LD_LIBRARY_PATH:.
- Execución: *a.out*

media.f90 e desviacion.*f90*

```
real function media(x, n)
   real, intent(in) ::  x(n)
   integer, intent(in) :: n
   media=0
   do i=1, n
       media=media+x(i)
   end do
   media=media/n
   return
   end function media
```

```
function desviacion(x, n, media)
real,intent(in):: x(n)
integer, intent(in) :: n
real, intent(in) :: media
desviacion=0
do i=1, n
    y = x(i) - media
    desviacion = desviacion + y*y
end do
desviacion = sqrt(desviacion/n)
return
end function desviacion
```

mediana.f90

```
real function mediana(x, n)
   real, intent(in) :: x(n)
   integer, intent(in) :: n
   call ordear(x, n)
   if(mod(n, 2) == 0) then
       mediana = (x(n/2)+x(n/2+1))/2
   else
       mediana = x(n/2+1)
   endif
   return
   end function mediana
```

ordea.f90

```
subroutine ordea(x, n)
real, intent(inout) :: x(n)
integer, intent(in) :: n
do i = 1, n
    vmin = x(i); imin = i
    do j = i + 1, n
        if(x(j) < vmin) then
            vmin = x(j); imin = j
        end if
    end do
    x(imin) = x(i); x(i) = vmin
end do
return
end subroutine ordear
```

principal.f90

```
program principal
 real,allocatable :: x(:)
 real :: media, mediana, m
 print *, "introduce n: "
 read *, n
 allocate(x(n))
 print *, "introduce x: "
 read *, x
 y = media(x, n)
 d = desviacion(x, n, y)
 m = mediana(x, n)
 call ordear(x, n)
 print *, "media= ", y, "desviacion= ", d, "mediana= ", m
 print *, "vector ordeado= ", x
 end program principal
```

Uso dunha libraría feita por outros (I)

- **Instalación** da libraría como administrador ou usuario. Dúas alternativas:
  - Paquetes precompilados (binarios), co synaptic,apt-get,rpm,...
  - Compilación do código fonte: *configure*, *make*, *make install* (como administrador)
- Coñecer o directorio no que se atopa o(s) arquivo(s) **libXX.a** ou **libXX.so**
- Acceder á **documentación** da libraría, para saber que subprogramas ten, os seus argumentos, valores retornados, etc.

Uso dunha libraría feita por outros (II)

- Dende o noso programa, chamamos aos subprogramas da libraría (ver documentación), pasándolle os argumentos axeitados (en número e tipo) e usando os valores retornados
- Coñecido o directorio do arquivo **libXX.a** ou **libXX.so**:

  f95 -Ldir programa.f90 -lXX

  dir: directorio no que *f95* debe buscar *libXX.a/libXX.so*

  -lXX: ***libXX.a*** é a librería que usará *programa.f90*

- Con librarías dinámicas:
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:dir
- Librarías libres en Fortran: http://www.fortran.com/tools.html
- Destacamos: Lapack, Linpack (http://www.netlib.org), Cernlib, Plplot, Libg2