Marcello Barnaba [415833] - Programmazione - Appello del 20/02/2002

Marcello Barnaba

Marcello Barnaba [415833] - Programmazione - Appello del 20/02/2002

Homepage ufficiale del programma: http://barnaba.openssl.it/

E-mail dell'autore: vjt@openssl.it

..:: GRCALC 1.0 ::..

Documentazione

· **Analisi/Specifica dei requisiti**

· **Funzioni Richieste al programma**

Il programma nasce come strumento per visualizzare in tempo reale grafici di equazioni matematiche in due dimensioni su un piano cartesiano ortogonale XY. Unico requisito di GRCALC e' un PC Intel o compatibile dotato di una scheda EGA/VGA, CGA o Hercules.

La funzione richiesta al programma e' di visualizzare a video il grafico di qualsiasi funzione rappresentabile su un piano cartesiano XY ed esprimibile in funzione di Y, quindi restano escluse tutte le equazioni in funzione di X, e qualsiasi equazione che non sia una corrispondenza biunivoca tra X ed Y: una funzione puo' associare ad un valore di X uno ed un solo valore di Y. Il programma permette la visualizzazione di tutte le curve geometriche (rette, parabole, iperboli, etc), funzioni goniometriche dirette (seno, coseno, tangente, cotangente) ed inverse (arcoseno, arcocoseno, arcotangente), della funzione logaritmo naturale, della funzione esponenziale (e^x) e della radice quadrata. E' possibile inoltre comporre tra di loro tutte queste funzioni.

· **I Dati di I/O**

· **INPUT**

L'immissione dei dati consiste in due parti: l'Immissione della funzione, e l'Immissione del fattore di zoom.

· Immissione della funzione

Il programma e' dotato di un'interfaccia testuale per l'inserimento della funzione, la quale va espressa nella normale notazione matematica, con la seguente tabella di corrispondenze per le funzioni:

```
sin ........... seno

cos ........... coseno

tan ........... tangente

cotan ......... cotangente

asin .......... arcoseno

acos .......... arcocoseno
```

```
 atan .......... arcotangente

 ln ............ logaritmo naturale

 e ............. esponenziale

 sqrt .......... radice quadra

 ---------------------------------

 tabella(1), FUNZIONI supportate
```

I seguenti operatori sono inoltre utilizzabili:

```
* ............. prodotto

 / ............. quoziente

 + ............. somma

 - ............. differenza

 ---------------------------------

 tabella(2), OPERATORI supportati
```

I dati vanno inseriti al prompt del programma, secondo questa sintassi:

ESPRESSIONE <invio>

ESPRESSIONE e‘ un‘equazione matematica, e puo‘ consistere in:

Una **FUNZIONE** cui va passata una **ESPRESSIONE** come argomento, un **VALORE** costante oppure una **VARIABILE**. inoltre e‘ possibile combinare una **ESPRESSIONE** in una sequenza di **ESPRESSIONI**, intervallate da **OPERATORI**.

Una **FUNZIONE** puo‘ esclusivamente essere una stringa appartenente alla prima colonna della tabella(1).

Un **VALORE** puo‘ essere una stringa di cifre (0..9) con lunghezza variabile, maggiore di uno. Per inserire valori negativi e‘ obbligatorio usare la notazione ”0 - **VALORE**”.

Una **VARIABILE** puo‘ esclusivamente essere il carattere alfanumerico ”**x**”, codice **ASCII** hex 0x78.

Ad una **FUNZIONE** e‘ obbligatorio far seguire una **ESPRESSIONE** come argomento.

Un **OPERATORE** deve essere preceduto e seguito da due **ESPRESSIONI**, e puo‘ essere solo uno dei caratteri ASCII appartenenti alla prima colonna della tabella(2).

Le **ESPRESSIONI** possono essere racchiuse in **PARENTESI**, quali ”(” [ascii 0x28] e ”)” [ascii 0x29], per migliorare la leggibilita‘ e per aumentare la precedenza dell‘espressione racchiusa rispetto a quella esterna.

Esempi:

```
 x ......................... funzione

 sin x ..................... funzione

 cos(e(x)) ................. funzione

 atan((1 - x) / (e(x * x))) . funzione
```

Per uscire dal programma e‘ sufficiente premere il tasto ”**INVIO**” al prompt di inserimento della funzione da disegnare.

· Immissione del fattore di zoom

Il fattore di zoom e‘ il numero di volte che la funzione verra‘ ingrandita a schermo, ed il valore immesso e‘ relativo esclusivamente alla risoluzione grafica che il programma utilizza durante l‘esecuzione. I valori vanno da 1 (1 pixel a video : 1 punto sul piano) a 1000 (1 pixel a video : 1000 punti sul piano).

· **OUTPUT**

Il programma presenta in output una videata divisa in 3 parti:

· Una barra superiore, la quale contiene la funzione visualizzata ed il fattore di zoom corrente.

· Una barra inferiore, la quale contiene la risoluzione attuale di visualizzazione.

· Un‘area centrale, in cui e‘ rappresentato il piano cartesiano ortogonale XY, dotata di una barra di progresso in basso. Il colore di questa e‘ determinato dall‘andamento della funzione mentre viene disegnata: ci sono 3 possibili situazioni, ognuna identificata da un colore diverso:

· **BLU** se la funzione e‘ definita nel punto in cui sta venendo disegnata, ed il valore di Y in quel punto e‘ visualizzabile a schermo nei valori di risoluzione e zoom correnti.

· **ROSSO** se la funzione e‘ definita in quel punto ma va oltre i limiti dello schermo e quindi non e‘ visualizzabile

· **GRIGIO** se la funzione non e‘ definita in quel punto.

· **Alcuni esempi di visualizzazione dei risultati**

```
y = cos(e(x)). [zoom 60x]
```

```
y = ln(cos(x*x)) + atan(x). [zoom 90x]
```

· **Progetto**

· **Descrizione in linguaggio naturale.**

La principale difficolta‘ incontrata nella stesura di un programma di questo tipo risiede nella situazione di non conoscenza della funzione da rappresentare, in quanto essa e‘ data in input dall‘utente e non e‘ hardcodata nel codice. C‘e‘ quindi la necessita‘ di trasformare questi caratteri ASCII passati in input dall‘utente in una forma che l‘elaboratore possa rapidamente interpretare e calcolare al momento del disegno del grafico. Questo processo di trasformazione e‘ racchiuso nel parser del programma, che ha come funzione madre **calc_parse_func**. Il tipo di dati strutturato scelto per la rappresentazione interna della funzione da disegnare e‘ una lista collegata di record **calc_term_t** tramite puntatori next, allocati dinamicamente durante l‘esecuzione, in cui ogni elemento della lista e‘ un termine della funzione. Il primo elemento e‘ contraddistinto dall‘avere il proprio indirizzo immagazzinato nella memoria referenziata dalla variabile globale **calc_term**, l‘ultimo e‘ contraddistinto dall‘avere il puntatore next settato a **nil**.

Un termine puo‘ essere di 6 tipi: **Numero**, **Variabile**, **Operatore**, **Funzione**, **Parentesi aperta**, e **Parentesi chiusa**: ogni tipo subisce un trattamento diverso durante il disegno della funzione; la struttura **calc_term_t** contiene tutti i campi possibili per l‘archiviazione dei 6 tipi. Piu‘ dettagliatamente:

· Term_type :: tipo del termine che stiamo considerando.

· Term_value :: valore eventuale del termine.

· Term_func_handler :: puntatore a funzione per funz. matematica.

· Term_oper_handler :: puntatore a funzione per operatore.

· Term_next :: puntatore al termine seguente.

Il valore di Term_next dipende sempre e comunque dalla posizione del termine nella lista.

Nel caso di un **Numero**, la struttura contiene i 2 puntatori a funzione nulli, **term_type** settato a CALC_TERM_TYPE_NUMBER, costante principale spiegata nel paragrafo della definizione di tipi e costanti, e **term_value** settato al valore intero del numero passato in input dall'utente.

Nel caso di una **Variabile**, la struttura contiene i 2 puntatori a funzione nulli, il tipo settato a CALC_TERM_TYPE_VARIABLE e **term_value** settato a 0.

Nel caso di un **Operatore**, la struttura contiene **term_func_handler** settato a nil, **term_type** settato a CALC_TERM_TYPE_OPERATOR, **term_value** settato a 0 e **term_oper_handler** settato all'indirizzo della funzione gestore dell'operatore considerato.

Nel caso di una **Funzione**, la struttura contiene **term_oper_handler** settato a nil, **term_type** settato a CALC_TERM_TYPE_FUNCTION, **term_value** settato a 0 e **term_func_handler** settato all'indirizzo della funzione gestore della funzione matematica considerata.

Nel caso di una **Parentesi aperta**, la struttura contiene i 2 puntatori a funzione nulli, il tipo settato a CALC_TERM_TYPE_BRACKETOPEN e **term_value** settato a 0.

Nel caso di una **Parentesi chiusa**, la struttura contiene i 2 puntatori a funzione nulli, il tipo settato a CALC_TERM_TYPE_BRACKETCLOSE e **term_value** settato a 0.

Il parsing genera la lista collegata secondo i criteri sopra spiegati. In caso di insuccesso, viene rilasciata la memoria allocata e viene generato un errore, altrimenti in caso di successo il controllo passa alle funzioni che poi disegneranno il grafico a video. Il disegno del grafico e' un'operazione basata su due concetti fondamentali: il piano cartesiano disegnato e' una traslazione ed uno zoom del piano dei pixel visualizzati a schermo, con origine nel centro del viewport assegnato al grafico, il quale viene disegnato scorrendo i pixel dall'estremo sinistro del viewport e calcolando per ogni pixel cui e' assegnato il valore di X in quel punto il corrispondente valore di Y e stampando il pixel a video, operando l'opportuna traslazione. Non e' presente alcun algoritmo di interpolazione, quindi si otterranno scarsi risultati nella visualizzazione

---

Durante il disegno, il motore scorre per ogni valore di X la lista di strutture create dal parser, e, a seconda del tipo, esegue le operazioni necessarie. Il punto in cui ci si trova e' identificato da un puntatore **p** che punta al termine corrente che si sta analizzando. In caso di funzione o operatore, chiama la funzione puntata nella struttura, in caso di valore, conserva in una variabile temporanea il numero contenuto nel **term_value** corrente, in caso di variabile assegna alla variabile il valore corrente di X. Il motore di disegno non fa alcun controllo sulla sintassi, in quanto e' compito del parser. Un termine di tipo **Parentesi aperta** causa la chiamata ricorsiva della stessa funzione di calcolo del valore di Y, ed un termine di tipo **Parentesi chiusa** causa l'uscita dalla chiamata ricorsiva. Le chiamate ricorsive modifi cano il valore di **p**, in modo da non alterare il flusso principale delle operazioni e da saltare il termine o i termini in parentesi per non processarli piu' volte.

Il programma e' facilmente estendibile, l'aggiunta di nuove funzioni e' immediata, ed inoltre e' possibile, con poche modifiche al codice, allocare dinamicamente piu' liste **calc_term_t** in modo da visualizzare piu' funzioni a video, poiche' al momento tutte le funzioni di disegno fanno riferimento alla **calc_term** globale.

· **Diagrammi di flusso**

Data la complessita' del problema da risolvere, vengono presentati esclusivamente i diagrammi di flusso del MAIN e della funzione calc_parse_func.

· **Definizione dei tipi e delle costanti principali.**

:: definizione dei tipi ::

· Calc_func_t :: string

· Calc_error_t :: short integer

· Calc_term_type_t :: short integer :: codice che individual un tipo termine.

· Calc_term_value_t :: real :: valore di un tipo termine.

· Calc_func_handler_t :: function(x : real) : real :: puntatore a funzione per le funzioni matematiche, la funzione puntata accetta un argomento real e ritorna un real.

· Calc_oper_handler_t :: function(x, y : real) : real :: puntatore a funzione per gli operatori matematici, la funzione puntata accetta due argomento real e ritorna un real.

· Calc_term_t_p :: puntatore ad un record calc_term_t.

· Calc_term_t :: record ::

- Term_type :: calc_term_type_t
- Term_value :: calc_term_value_t
- Term_func_handler :: calc_func_handler_t
- Term_oper_handler :: calc_oper_handler_t
- Term_next :: calc_term_t_p

· Calc_parse_state_t :: shortint :: stato interno del parser.

:: costanti principali ::

· CALC_OPS :: equivale a **4**

· CALC_FUNX :: equivale a **10**

· Calc_oper_table :: array di calc_oper_handler_t, formato da CALC_OPS elementi

· Calc_func_table :: array di record

- Func_name :: stringa di max 5 caratteri :: nome della funzione
- Func_handler :: calc_func_handler_t :: gestore della funzione

Formato da CALC_FUNX elementi.

· CALC_TERM_TYPE_NUMBER = 1 :: tipo termine NUMERO

· CALC_TERM_TYPE_FUNCTION = 2 :: tipo termine FUNZIONE

· CALC_TERM_TYPE_OPERATOR = 3 :: tipo termine OPERATORE

· CALC_TERM_TYPE_VARIABLE = 4 :: tipo termine VARIABILE

· CALC_TERM_TYPE_BRACKETOPEN = 5 :: tipo termine PARENTESI APERTA

· CALC_TERM_TYPE_BRACKETCLOSE = 6 :: tipo termine PARENTESI CHIUSA

· **Descrizione delle componenti del programma (unita' principale, procedure e funzioni).**

:: Unita' principale ::

Obiettivo: avvio ed inizializzazione del programma, ed eventuale uscita prematura da esso in caso di non disponibilita' di hardware grafico.

Parametri: nessuno.

Valore di ritorno: ExitCode predefinito.

Variabili (globali):

· calc_func :: calc_func_t :: stringa funzione come passata in input dall'utente.

· calc_term :: calc_term_t_p :: puntatore al primo elemento della lisca collegata dei termini della funzione.

· calc_errno :: calc_error_t :: codice di errore corrente del programma.

· calc_zoom :: intero :: valore di zoom corrente.

· maxx, maxy :: word :: x ed y massimi nella visualizzazione corrente.

:: PARSER ::

:: calc_parse_func ::

Obiettivo: parsare una funzione passata come stringa e trasformarla in una lista concatenata di strutture **calc_term_t**.

Parametri :

· f :: calc_func_t :: stringa da parsare

Variabili:

· i :: short integer :: contatore principale.

· done :: booleano :: flag che stabilisce se abbiamo terminato il parsing o meno.

· p :: calc_term_t_p :: puntatore temporaneo per allocare gli elementi della lista collegata di termini.

· state :: calc_parse_state_t stato interno del parser.

· token_buf :: stringa ::buffer temporaneo per conservare parti della stringa da parsare per poi processarli in seguito.

· ret :: booleano :: valore di ritorno della funzione che poi verra‘ ritornato al chiamante (adottato data la ricorsivita‘ della funzione).

Valore di ritorno: boolean :: true in caso di successo, false in caso di insuccesso.

:: sottoprocedure ::

:: fail ::

Obiettivo: segnalare un errore di sintassi e settare il valore di ritorno della funzione a false.

Parametri: nessuno.

Variabili: nessuna.

Valore di ritorno: nessuno.

:: func2id ::

Obiettivo: cercare nella tabella delle funzioni una stringa di caratteri corrispondente a quella passata come parametro, e ritornare il numero d‘ordine della funzione cercata, in modo da poter prendere il puntatore alla funzione gestore di quella funz. matematica e scriverlo nella struttura termine corrente.

Parametri: s :: stringa :: stringa da cercare.

Variabili: i :: short integer :: contatore per la ricerca lineare all‘interno della tabella di funzioni.

Valore di ritorno: short integer :: 0 in caso di insuccesso, un numero n con: 1 < n < CALC_FUNX in caso di successo.

:: changestate ::

Obiettivo: cambiare lo stato interno del parser e salvare nel termine corrente le informazioni relative ad esso, ed allocare un nuovo termine.

Parametri: newstate :: calc_parse_state_t :: nuovo stato in cui transitare.

Variabili:

· good_syntax :: booleano :: flag il cui valore indica se la sintassi e' corretta o meno.

· id :: short integer :: indice nella tabella delle funzioni relativo alla funzione che si sta parsando.

· val_code :: intero :: valore di ritorno della val() che viene eseguita all'interno della procedura.

Valore di ritorno: nessuno.

:: check_brackets ::

Obiettivo: verificare il bilanciamento delle parentesi.

Parametri: nessuno.

Variabili:

· i :: shortint :: contatore per la ricerca binaria all'interno della stringa.

· j :: shortint :: indicatore dello stato delle parentesi. E' maggiore di zero se ci sono piu' parentesi aperte che chiuse, minore di zero se ci sono piu' parentesi chiuse che aperte, uguale a zero se le parentesi sono bilanciate.

· len :: shortint :: lunghezza della stringa passata in input.

Valore di ritorno: true se le parentesi sono bilanciate, false in caso contrario.

:: MOTORE DI DISEGNO ::

:: get_y_value ::

Obiettivo: calcolare il valore di Y dato X e la funzione da disegnare, passata sotto forma di puntatore ad una lista di strutture **calc_term**.

Parametri:

· x :: reale :: valore di X per cui calcolare la Y

· p :: calc_term_t_p :: puntatore al punto in cui si vuole evalutare la funzione, [passato per riferimento]

Variabili:

· current_value :: calc_term_value_t :: valore corrente nell'evalutazione.

· current_oper :: calc_oper_handler_t :: operatore corrente.

· currently_operating :: booleano :: flag che ci dice se stiamo eseguendo una operazione con un operatore binario o meno.

Valore di ritorno: real :: valore della Y corrispondente al dato valore della X

:: sottoprocedure ::

:: evaluate_func ::

Obiettivo: evalutare il valore di una funzione matematica o di una successione di esse [ad es. sin(cos(tan(x)))].

Parametri: nessuno.

Variabili:

· current_func :: calc_func_handler_t :: funzione corrente.

· ret :: calc_term_value_t :: valore di ritorno della funzione (adottato data la ricorsivita‘ della funzione).

Valore di ritorno: calc_term_value_t :: valore della Y corrispondente al valore X specificato dal **p** corrente.

:: draw_carthesio ::

Obiettivo: Disegnare un piano cartesiano ortogonale nel viewport corrente, e stampare a video i tick i quali indicano i numeri relativi corrispondenti ai punti disegnati sul piano.

Parametri: nessuno.

Variabili:

· vp :: viewporttype :: viewport corrente di visualizzazione, per i suoi dettagli, consultare l‘help in linea del Turbo Pascal 7.0.

· relmaxx, relmaxy :: word :: X ed Y massimi relativi al viewport corrente.

· str :: stringa :: buffer per stampare i valori della X e della Y sugli assi del piano.

· th, tw :: interi :: contengono l‘altezza e l‘ampiezza dei valori che vengono stampati sotto gli assi.

· ts :: textsettigstype :: record che contiene i settaggi del testo quando la funzione viene chiamata. Per i suoi dettagli, consultare l‘help in linea del Turbo Pascal 7.0.

· increment :: intero :: valore relativo allo zoom di visualizzazione che contiene l‘incremento tra un valore ed il suo successivo durante la stampa sul piano.

· i :: intero :: contiene il valore stesso da stampare sotto il tick sugli assi.

· x, y :: interi :: contengono di volta in volta nel ciclo la posizione in cui stiamo stampando il valore ed il tick sull‘asse del piano.

Valore di ritorno: nessuno.

:: draw_graph ::

Obiettivo: Stampare a video il grafico.

Parametri: nessuno.

Variabili:

· x, y :: interi :: coordinate relative al piano XY rappresentato.

· vp :: viewporttype :: viewport corrente di visualizzazione, per i suoi dettagli, consultare l‘help in linea del Turbo Pascal 7.0.

· relmaxx, relmaxy :: word :: X ed Y massimi relativi al viewport corrente.

· barcolor :: shortint :: colore della barra di progresso.

· p :: calc_term_t_p :: puntatore che scorre la lista di termini per calcolare i valori di Y in funzione di X**.**

Valore di ritorno: nessuno.

:: calc_graph_main ::

Obiettivo: chiamare le altre funzioni per la stampa del grafico, stampa delle barre di stato e preparazione del viewport per il piano cartesiano.

Parametri: nessuno.

Variabili: bar_width :: intero :: spessore della barra di stato, calcolato relativamente all'altezza del font utilizzato.

Valore di ritorno: nessuno.

:: FUNZIONI DI UTILITA' VARIE ::

:: calc_perror ::

Obiettivo: stampare a video la stringa corrispondente al codice di errore corrente.

Parametri: err :: calc_error_t :: codice errore da visualizzare.

Variabili: nessuna.

Valore di ritorno: nessuno.

:: cleanup ::

Obiettivo: liberare la memoria occupata dalla lista di strutture termine.

Parametri: nessuno.

Variabili: p, q :: calc_term_t_p :: puntatori per lo scorrimento e liberazione della memoria.

Valore di ritorno: nessuno.

:: int2str ::

Obiettivo: convertire un tipo di dato intero in una string.

Parametri: i :: intero :: intero da convertire.

Variabili: buf :: string :: stringa di appoggio temporaneo.

Valore di ritorno: nessuno

:: draw_bar ::

Obiettivo: disegnare una barra rettangolare a video con dentro del testo.

Parametri:

· x1, y1, x2, y2 :: word :: vertici della diagonale della barra rettangolare, con x1 e y1 >= x2 e y2.

· str :: string :: testo da stampare nella barra.

Variabili: nessuna.

Valore di ritorno: nessuno.

:: calc_setgraph ::

Obiettivo: verificare la presenza di hardware grafico e, in caso positivo, inizializzare la modalita' grafica. Dopo l'inizializzazione e' possibile utilizzare la funzione per tornare in modalita' terminale e successivamente ripassare in modalita' grafica.

Parametri: todo :: short integer :: cosa deve fare la funzione.

Variabili:

· gd :: intero :: codice del driver grafico da utilizzare.

· gml, gmh :: interi :: codici rispettivamente della modalita' grafica piu' bassa e piu' alta disponibile.

Valore di ritorno: boolean :: nel caso in cui la funzione debba inizializzare la modalita' grafica, il valore indica successo con true, insuccesso con false. Negli altri casi, il valore di ritorno e' indefinito.

:: calc_read_data ::

Obiettivo: leggere i dati in input dall'utente: la funzione e il valore di zoom. Impostare il valore di zoom. Controllare se il valore di zoom inserito e' corretto. Verificare se l'utente abbia digitato la sequenza che causa l'uscita dal programma.

Parametri: nessuno.

Variabili:

· buf :: string :: buffer di appoggio per la val().

· val_code :: integer :: valore di ritorno per la val().

Valore di ritorno: boolean :: true nel caso in cui sia stata letto un valore di zoom corretto ed una stringa che rappresenta la funzione matematica. false se e' stata inserita la sequenza di tasti che causa l'uscita dal programma.

:: calc_banner ::

Obiettivo: stampare un simpatico banner che mostra il nome e la versione del software :).

Parametri: nessuno.

Variabili: nessuno.

Valore di ritorno: nessuno.

:: calc_exitproc ::

Obiettivo: procedura di uscita dal porogramma, stampa solo alcune righe di ringraziamento ed attende un input qualsiasi dall'utente per uscire.

Parametri: nessuno.

Variabili: nessuno.

Valore di ritorno: nessuno.

· **Situazioni di errore che il programma puo' gestire.**

Per ognuna di queste situazioni, viene visualizzato un messaggio di errore.

· Assenza di Hardware Grafico o mancata inizializzazione della modalita' grafica.

· Errata sintassi della funzione inserita.

· Valore di zoom inserito non valido.

· **Wish-list per la versione successiva (se mai uscira')**

· Un algoritmo di interpolazione per le funzioni che variano troppo rapidamente.

· Visualizzazione di due o piu' grafici.

· Visualizzazione a video dei punti di intersezione con gli assi e tra gli eventuali grafici multipli.

· Salvataggio su .bmp o .pcx del grafico ottenuto.

· Portabilita' del sorgente, non basarsi sulle funzioni della BGI ma utilizzare dei wrapper contenuti in un include separato.

· Trasposizione in linguaggio C o C++, date alcune mancanze del PASCAL: in primo luogo degli operatori bit-a-bit, in secondo luogo per la gestione dei puntatori ed in terzo luogo per la maggiore portabilita' del codice C su multiple piattaforme dove non sono disponibili compilatori PASCAL.

· **Codifica (in linguaggio PASCAL).**

```
{ ************************GRCALC************************ }
{
  Copyright (C) 2002 Marcello Barnaba <vjt@openssl.it>,
  licenziato secondo i termini della GNU GPL versione 2
  (www.gnu.org/gpl.html) o, ad opzione dell`utente,
  qualsiasi versione successiva.
  programma per visualizzare una semplice funzione matematica
  a schermo, ispirato dalla `Calcolatrice Grafica' del MacOS
  e da GNUplot (http://www.gnu.org/software/gnuplot).
}
program graphic_calc;
{ direttive per il coprocessore matematico. }
{IFDEF CPU87}
{$N+;$E-}
{ELSE}
{$N-;$E+}
{ENDIF}
{ disabilita l`inserimento nell`eseguibile dei simboli di debug. }
{$D-}
{ crt per clrscr, graph per la totalita` delle funzioni grafiche }
uses crt, graph;
{ linka i driver video nell`eseguibile, in modo da non essere
  costretti a distribuire il .BGI assieme al .EXE. }
procedure EGAVGADriver; external;
procedure CGADriver; external;
procedure HERCDriver; external;
{$L egavga.obj}
{$L cga.obj}
{$L herc.obj}
{ dichiarazioni forward per le funzioni da inserire nelle tabelle
  const. }
{$F+}
function calc_add(x, y : real) : real; forward;
function calc_sub(x, y : real) : real; forward;
function calc_mul(x, y : real) : real; forward;
function calc_div(x, y : real) : real; forward;
```

```
function calc_sin(x : real) : real; forward;
function calc_cos(x : real) : real; forward;
function calc_tan(x : real) : real; forward;
function calc_cotan(x : real) : real; forward;
function calc_asin(x : real) : real; forward;
function calc_acos(x : real) : real; forward;
function calc_atan(x : real) : real; forward;
function calc_ln(x : real) : real; forward;
function calc_exp(x : real) : real; forward;
function calc_sqrt(x : real) : real; forward;
{$F-}
type
 calc_func_t = string; { tipo della stringa che contiene la funzione
        inserita dall`utente }
 calc_error_t = shortint; { tipo del codice di errore gestito dalla
      calc_perror() }
 calc_term_type_t = shortint; { tipo del codice che identifica un
          tipo di termine all`interno
     della struttura 'calc_term_t' }
 calc_term_value_t = real; { tipo del valore di un termine }
 { tipo puntatore a funzione per le funzioni matematiche }
 calc_func_handler_t = function(x : real) : real;
 { tipo puntatore a funzione per gli operatori }
 calc_oper_handler_t = function(x, y : real) : real;
 calc_term_t_p = ^calc_term_t; { puntatore ad una struttura calc_term,
       per la creazione della lista collegata }
 calc_term_t = record { struttura termine, pesante in termini di memoria,
       poiche` contiene tutti i possibili elementi che
       possono identificare un termine: sia il valore,
       che un operatore non possiede, sia due puntatori
       a funzione, per le funzioni e gli operatori, ed
       un puntatore alla struttura successiva.
       alternativamente sarebbe stato possibile
       utilizzare un oggetto base e poi utilizzare
       l`incapsulamento e l`ereditarieta` per creare
       strutture polimorfiche, ma a mio parere il
```

```
      programma sarebbe diventato troppo complesso
      e pesante di quanto non lo sia gia'. }
  term_type : calc_term_type_t; { codice tipo del termine }
  term_value : calc_term_value_t; { valore del termine, nel
             caso in cui questo termine sia
      un numero }
  term_func_handler : calc_func_handler_t; { puntatore a funzione per
            funzione matematica }
  term_oper_handler : calc_oper_handler_t; { puntatore a funzione per
               operatore }
  term_next : calc_term_t_p;               { elemento successivo, nil
          in caso di
         ultimo elem. }
 end;
 calc_parse_state_t = shortint; { tipo dello stato interno del parser }
const
 { versione. }
 CALC_VERSION = '1.0';
 { costanti per la gestione degli errori. }
 CALC_NO_ERROR = 1;
 CALC_ERROR_INITGRAPH = 2;
 CALC_ERROR_BADFUNC = 3;
 CALC_ERROR_BADINTEGER = 4;
 CALC_ERROR_FUNCTIONNOTDEFINED = 5;
 CALC_ERRORS = 5; { numero errori. }
 calc_errlist : array[1..CALC_ERRORS] of string = (
  'Nessun errore.',
  'Non sono riuscito ad inizializzare la modalita` grafica.',
  'Errore di sintassi nella funzione.',
  'Intero non valido.',
  'Funzione non definita in questo punto.'
 );
 { tipi di termini. }
 CALC_TERM_TYPE_NUMBER = 1;
 CALC_TERM_TYPE_FUNCTION = 2;
 CALC_TERM_TYPE_OPERATOR = 3;
```

```
CALC_TERM_TYPE_VARIABLE = 4;
CALC_TERM_TYPE_BRACKETOPEN = 5;
CALC_TERM_TYPE_BRACKETCLOSE = 6;
{ costanti per gli operatori. }
CALC_TERM_TYPE_OPERATOR_ADD = 1;
CALC_TERM_TYPE_OPERATOR_SUB = 2;
CALC_TERM_TYPE_OPERATOR_MUL = 3;
CALC_TERM_TYPE_OPERATOR_DIV = 4;
{ costanti che descrivono il numero di funzioni e operatori supportati. }
CALC_FUNX = 10;
CALC_OPS = 4;
{ stati del parser. }
CALC_PARSE_IDLE = 1;   { inizio della stringa. }
CALC_PARSE_READNUMBER = 2;  { il parser sta leggendo un numero. }
CALC_PARSE_READFUNCTION = 3; { il parser sta leggendo una funzione }
CALC_PARSE_READOPERATOR = 4; { il parser sta leggendo un operatore }
CALC_PARSE_READVARIABLE = 5; { il parser sta leggendo 'x' }
CALC_PARSE_BRACKETOPEN = 6;  { il parser sta leggendo '(' }
CALC_PARSE_BRACKETCLOSE = 7;     { il parser sta leggendo ')' }
{ settaggi grafici. }
CALC_GRAPH_BAR_SHIFT = 4;  { dimensione della barra relativa al
       font corrente. }
CALC_GRAPH_DEFAULT_ZOOM = 50; { fattore di zoom di default. }
CALC_GRAPH_LINE_COLOR = yellow; { colore della curva. }
CALC_GRAPH_DELAY = 100;           { tempo di inattivita` tra il disegno
       di un pixel ed un altro. }
CALC_GRAPH_CARTHESIO_TICK_HALF_LENGTH = 3; { meta` della dimensione di
          una stanghetta del piano
          cartesiano. }
CALC_GRAPH_PROGRESSBAR_COLOR_DEFINED = blue; { colore della barra
            inferiore quando la
            funz. e` definita. }
CALC_GRAPH_PROGRESSBAR_COLOR_NOTDEFINED = lightgray; { colore della barra
        inferiore quando la
        funz. non e` def. }
CALC_GRAPH_PROGRESSBAR_COLOR_OOB = red; { colore della barra inferiore
```

```
        quando la funz. e` definita ma
        va oltre i limiti dello schermo.
            }
 CALC_GRAPH_PROGRESSBAR_WIDTH = 8; { spessore della barra inferiore. }
 { costanti per calc_setgraph() }
 CALC_INIT = 0; { inizializza la mod. grafica. }
 CALC_CRT = 1;  { ripristina la mod. terminale. }
 CALC_GRAPH = 2;{ ripristina la mod. grafica. }
 CALC_CLOSE = 3;{ chiudi la mod. grafica. }
 { tabella degli operatori }
 calc_oper_table : array [1..CALC_OPS] of calc_oper_handler_t =
  (calc_add, calc_sub, calc_mul, calc_div);
 { tabella delle funzioni }
 calc_func_table : array [1..CALC_FUNX] of record { tipo della tabella
      delle funzioni supportate }
  func_name : string[5]; { nome della funzione }
  func_handler : calc_func_handler_t; { puntatore alla
           funzione gestore }
 end = (
  (func_name : 'sin'; func_handler : calc_sin),
  (func_name : 'cos'; func_handler : calc_cos),
  (func_name : 'tan'; func_handler : calc_tan),
  (func_name : 'asin'; func_handler : calc_asin),
  (func_name : 'acos'; func_handler : calc_acos),
  (func_name : 'atan'; func_handler : calc_atan),
  (func_name : 'ln'; func_handler : calc_ln),
  (func_name : 'e'; func_handler : calc_exp),
  (func_name : 'cotan'; func_handler : calc_cotan),
  (func_name : 'sqrt'; func_handler : calc_sqrt)
 );
var
 calc_func : calc_func_t; { stringa che contiene la funzione come scritta
     dall`utente. }
 calc_term : calc_term_t_p; { testa della lista collegata di termini. }
 calc_errno : calc_error_t; { codice errore attuale. }
```

```
 calc_zoom : integer; { valore di zoom nella visualizzazione corrente. }
 maxx, maxy : word; { x ed y massimi nella visualizzazione corrente. }
{ permetti le chiamate FAR per queste funzioni, in modo
  che possano essere chiamate tramite puntatori a funzione }
{$F+}
{ funzioni per il calcolo del valore delle operazioni base. }
function calc_add(x, y : real) : real;
begin calc_add := x + y; end;
function calc_sub(x, y : real) : real;
begin calc_sub := x - y; end;
function calc_mul(x, y : real) : real;
begin calc_mul := x * y; end;
function calc_div(x, y : real) : real;
begin
 calc_errno := CALC_ERROR_FUNCTIONNOTDEFINED;
 calc_div := 0;
 if y <> 0 then begin
  calc_div := x / y;
  calc_errno := CALC_NO_ERROR;
 end;
end;
{ funzioni per il calcolo del valore delle funz. matematiche. }
function calc_tan(x : real) : real;
begin calc_tan := sin(x) / cos(x); end;
function calc_cotan(x : real) : real;
var y : real;
begin
 calc_cotan := 0;
 y := sin(x);
 if y <> 0 then
  calc_cotan := cos(x) / y
 else
  calc_errno := CALC_ERROR_FUNCTIONNOTDEFINED;
end;
{ operazione di asin trovata sulla FAQ del newsgroup comp.lang.pascal }
```

```
function calc_asin(x : real) : real;
var _sqr, _sqrt : real;
begin
 _sqr := (1 - sqr(x));
 calc_errno := CALC_ERROR_FUNCTIONNOTDEFINED;
 calc_asin := 0;
 if _sqr > 0 then begin
  _sqrt := sqrt(_sqr);
  if _sqrt <> 0 then begin
   calc_asin := arctan( x / _sqrt );
   calc_errno := CALC_NO_ERROR;
  end;
 end;
end;
{ operazione di asin trovata sulla FAQ del newsgroup comp.lang.pascal }
function calc_acos(x : real) : real;
var _sqr : real;
begin
 calc_errno := CALC_ERROR_FUNCTIONNOTDEFINED;
 calc_acos := 0;
 if x <> 0 then begin
  _sqr := (1 - sqr(x));
  if _sqr > 0 then begin
   calc_acos := arctan(sqrt(_sqr) / x);
   calc_errno := CALC_NO_ERROR;
  end;
 end;
end;
function calc_sin(x : real) : real;
begin calc_sin := sin(x); end;
function calc_cos(x : real) : real;
begin calc_cos := cos(x); end;
function calc_atan(x : real) : real;
begin calc_atan := arctan(x); end;
function calc_exp(x : real) : real;
begin calc_exp := exp(x); end;
```

```
function calc_ln(x : real) : real;
begin
 calc_ln := 0;
 if x > 0 then
  calc_ln := ln(x)
 else
  calc_errno := CALC_ERROR_FUNCTIONNOTDEFINED;
end;
function calc_sqrt(x : real) : real;
begin
 calc_sqrt := 0;
 if x > 0 then
  calc_sqrt := sqrt(x)
 else
  calc_errno := CALC_ERROR_FUNCTIONNOTDEFINED;
end;
{$F-}
{ procedura che stampa a schermo la descrizione
  di un errore identificato da un codice, ispirata
  alla perror(3) C ISO. ho scelto di passare l`errno
  come parametro poiche` non ci sono moltissimi errori
  da gestire in questo programma. }
procedure calc_perror(err : calc_error_t);
begin
 writeln('Errore: ', calc_errlist[err]);
end;
{ procedura per il rilascio della memoria occupata
  dalla lista dei termini della funzione. }
procedure cleanup;
var p, q : calc_term_t_p;
begin
 p := calc_term;
 while p <> nil do begin
  q := p^.term_next;
  dispose(p);
  p := q;
```

```
 end;
end;
function calc_parse_func(f : calc_func_t) : boolean;
var i : shortint; { contatore principale }
var done : boolean; { abbiamo terminato il parsing ? }
var p : calc_term_t_p; { puntatore temporaneo per allocare gli elementi
        della lista }
var state : calc_parse_state_t; { stato interno del parser }
var token_buf : string; { buffer temporaneo per conservare pezzi di stringa
    (come 'sin', 'cos') per poi processarli in seguito
    con la func2id() }
var wasspace : boolean; { bool di utilita` che indica se il precedente char
        era uno spazio }
var ret : boolean;
 { setta il fallimento del parsing e conseguente ritorno false
   della procedura genitore }
 procedure fail;
 begin
  done := true;
  ret := false;
 end;
 { prende in input un nome di funzione matematica (sin, cos, tan . .)
   e ritorna un ID da inserire in una struttura calc_term_t }
 function func2id(s : string) : shortint;
 var i : shortint;
 begin
  func2id := 0;
  for i := 1 to CALC_FUNX do begin
   if s = calc_func_table[i].func_name then begin
    func2id := i;
    exit;
   end;
  end;
 end;
 { cambia lo stato interno del parser, effettuando gli
   opportuni controlli di sintassi e salvando i dati
```

```
  dello stato precedente nella struttura correntemente
  puntata da p, e alloca il successore di p facendolo
  puntare nella p corrente a term_next }
procedure changestate(newstate : calc_parse_state_t);
 { alloca un nuovo elemento della lista, e aggiorna
   il valore corrente di p a questo nuovo elemento }
 procedure new_p;
 begin
  new(p^.term_next);
  p := p^.term_next;
  p^.term_next := nil;
 end;
var val_code : integer; { valore di ritorno della val() }
var good_syntax : boolean;
var id : shortint;
begin
 { controllo sintassi. }
 good_syntax := false;
 case newstate of
  CALC_PARSE_READNUMBER : good_syntax :=
     (state = CALC_PARSE_READFUNCTION) or
     (state = CALC_PARSE_READOPERATOR) or
     (state = CALC_PARSE_BRACKETOPEN) or
     (state = CALC_PARSE_IDLE);
  CALC_PARSE_READOPERATOR : good_syntax :=
     (state = CALC_PARSE_READNUMBER) or
     (state = CALC_PARSE_BRACKETCLOSE) or
     (state = CALC_PARSE_READVARIABLE);
  CALC_PARSE_READFUNCTION : good_syntax :=
     (state = CALC_PARSE_IDLE) or
     (state = CALC_PARSE_READOPERATOR) or
     (state = CALC_PARSE_READFUNCTION) or
     (state = CALC_PARSE_BRACKETOPEN);
  CALC_PARSE_READVARIABLE : good_syntax :=
```

```
    (state = CALC_PARSE_IDLE) or
    (state = CALC_PARSE_READOPERATOR) or
    (state = CALC_PARSE_READFUNCTION) or
    (state = CALC_PARSE_BRACKETOPEN);
 CALC_PARSE_BRACKETOPEN : good_syntax :=
    (state = CALC_PARSE_IDLE) or
    (state = CALC_PARSE_READFUNCTION) or
    (state = CALC_PARSE_READOPERATOR) or
    (state = CALC_PARSE_BRACKETOPEN);
 CALC_PARSE_BRACKETCLOSE : good_syntax :=
    (state = CALC_PARSE_READNUMBER) or
    (state = CALC_PARSE_READVARIABLE) or
    (state = CALC_PARSE_BRACKETCLOSE);
 CALC_PARSE_IDLE : begin
      good_syntax :=
     (state = CALC_PARSE_READVARIABLE) or
     (state = CALC_PARSE_BRACKETCLOSE) or
     (state = CALC_PARSE_READNUMBER);
    done := true;
     end;
end;
{ salvataggio termine corrente }
if good_syntax then begin
 case state of
  CALC_PARSE_READNUMBER : begin
   new_p;
   p^.term_type := CALC_TERM_TYPE_NUMBER;
   val(token_buf, p^.term_value, val_code);
   if val_code = 0 then begin
    p^.term_func_handler := nil;
    p^.term_oper_handler := nil;
    token_buf := '';
    state := newstate;
   end
   else
    fail;
```

```
end;
CALC_PARSE_READFUNCTION : begin
 id := func2id(token_buf);
 if id <> 0 then begin
  new_p;
  p^.term_type := CALC_TERM_TYPE_FUNCTION;
  p^.term_value := 0;
  p^.term_oper_handler := nil;
  p^.term_func_handler :=
   calc_func_table[id].func_handler;
  token_buf := '';
  state := newstate;
 end
 else
  fail;
end;
CALC_PARSE_READOPERATOR : begin
 new_p;
 p^.term_type := CALC_TERM_TYPE_OPERATOR;
 p^.term_value := 0;
 p^.term_func_handler := nil;
 case token_buf[1] of
  '*' : p^.term_oper_handler :=
  calc_oper_table[CALC_TERM_TYPE_OPERATOR_MUL];
  '+' : p^.term_oper_handler :=
  calc_oper_table[CALC_TERM_TYPE_OPERATOR_ADD];
  '/' : p^.term_oper_handler :=
  calc_oper_table[CALC_TERM_TYPE_OPERATOR_DIV];
  '-' : p^.term_oper_handler :=
  calc_oper_table[CALC_TERM_TYPE_OPERATOR_SUB];
 end;
 token_buf := '';
 state := newstate;
end;
CALC_PARSE_READVARIABLE : begin
 new_p;
```

```
    p^.term_type := CALC_TERM_TYPE_VARIABLE;
    p^.term_value := 0;
    p^.term_oper_handler := nil;
    p^.term_func_handler := nil;
    state := newstate;
   end;
   CALC_PARSE_BRACKETOPEN : begin
    new_p;
    p^.term_type := CALC_TERM_TYPE_BRACKETOPEN;
    p^.term_value := 0;
    p^.term_oper_handler := nil;
    p^.term_func_handler := nil;
    state := newstate;
   end;
   CALC_PARSE_BRACKETCLOSE : begin
    new_p;
    p^.term_type := CALC_TERM_TYPE_BRACKETCLOSE;
    p^.term_value := 0;
    p^.term_oper_handler := nil;
    p^.term_func_handler := nil;
    state := newstate;
   end;
   CALC_PARSE_IDLE : state := newstate;
  end; { fine salvataggio termine corrente }
 end { fine controllo sintassi }
 else
  fail;
end;
{ verifica che le parentesi siano bilanciate }
function check_brackets : boolean;
var len : shortint;
var i : shortint;
var j : shortint;
begin
 len := length(f);
```

```
  j := 0;
  check_brackets := false;
  for i := 1 to len do begin
   if f[i] = '(' then
    inc(j)
   else if f[i] = ')' then
    dec(j);
  end;
  if j = 0 then
   check_brackets := true;
 end;
 { fine sottoprocedure }
begin
 { metti a 0 il char successivo alla lunghezza della stringa. }
 f[length(f) + 1] := #0;
 { inizializza lo stato interno del parser. }
 state := CALC_PARSE_IDLE;
 { inizializza il contatore principale e
   il buffer che conterra` pezzi della stringa
   in input per processarli in seguito. }
 i := 0;
 token_buf := '';
 { inizializza i puntatori e la lista collegata,
   creando la testa che poi andra` tagliata.
   questo per alleggerire l`algoritmo di aggiungimento
   di elementi alla lista. se non venisse allocata una testa
   subito, poiche` c`e` l`esigenza di conservare l`indirizzo
   del primo elemento della lista per poi processarla in seguito
   e scorrerla, quando verra` visualizzata la funzione, bisognerebbe
   inserire una struttura di selezione binaria (if) nel ciclo
   della procedura changestate, in modo che se la testa della lista
   non e` ancora definita (e` nil), l`elemento che viene allocato
   diventa la testa. allocando prima una testa fittizia, abbiamo la
   certezza che durante il parsing il primo elemento e` gia` allocato,
   risparmiando il controllo, e dovremo solo aggiungere nuovi elementi
   alla lista. alla fine del parsing, se la sintassi e` corretta viene
```

```
  tagliata la testa e calc_term viene fatto puntare al secondo elemento
  della lista, che diventa il primo e la lista e` allocata
  correttamente. }
new(p);
calc_term := p; { testa. }
p^.term_next := nil;
{ presumi che la funzione abbia sintassi corretta. }
ret := true;
{ inizia ad iterare. }
wasspace := false;
done := false;
if not check_brackets then
 fail;
while not done do begin
 inc(i);
 case f[i] of
  'x' : changestate(CALC_PARSE_READVARIABLE);
  '0'..'9' :
   begin
    if state <> CALC_PARSE_READNUMBER then
     changestate(CALC_PARSE_READNUMBER);
    token_buf := token_buf + f[i];
   end;
  '*', '+',
  '/', '-':
   begin
    changestate(CALC_PARSE_READOPERATOR);
    token_buf := token_buf + f[i];
   end;
  'a'..'z' :
   begin
    if (state <> CALC_PARSE_READFUNCTION) or
     ((state = CALC_PARSE_READFUNCTION)
       and wasspace) then begin
     changestate(CALC_PARSE_READFUNCTION);
```

```
      wasspace := false;
     end;
     token_buf := token_buf + f[i];
    end;
   '(' : changestate(CALC_PARSE_BRACKETOPEN);
   ')' : changestate(CALC_PARSE_BRACKETCLOSE);
   ' ' : wasspace := true;
   #0 : changestate(CALC_PARSE_IDLE);
   else
    fail;
  end;
 end;
 { taglia via la testa della lista. }
 p := calc_term^.term_next;
 dispose(calc_term);
 calc_term := p;
 if not ret then begin
  calc_errno := CALC_ERROR_BADFUNC;
  cleanup;
 end;
 calc_parse_func := ret;
end;
{ converti un intero in una stringa. }
function int2str(i : integer) : string;
var buf : string;
begin
 str(i, buf);
 int2str := buf;
end;
{ cuore del disegno del grafico: procedura che
  a seconda del valore x che gli viene passato,
  ritorna il valore corrispondente della y,
  scorrendo la lista dei termini ed effettuando
  le operazioni necessarie }
```

```
function get_y_value(x : real; var p : calc_term_t_p) : real;
var current_value : calc_term_value_t;
var current_oper : calc_oper_handler_t;
var currently_operating : boolean;
{ funzione ricorsiva per evalutare il valore di
  una singola o di una serie di funzioni (es. sin cos tan x)
  matematiche, partendo dall`elemento che gli viene passato
  tramite puntatore al termine corrente. }
function evaluate_func : calc_term_value_t;
var current_func : calc_func_handler_t;
var ret : calc_term_value_t;
begin
 current_func := p^.term_func_handler;
 p := p^.term_next;
 case p^.term_type of
  CALC_TERM_TYPE_FUNCTION :
   ret := current_func(evaluate_func);
  CALC_TERM_TYPE_VARIABLE :
   ret := current_func(x);
  CALC_TERM_TYPE_NUMBER :
   ret := current_func(p^.term_next^.term_value);
  CALC_TERM_TYPE_BRACKETOPEN :
   begin
    p := p^.term_next;
    ret := current_func(get_y_value(x, p));
   end;
 end;
 evaluate_func := ret;
end;
begin
 current_value := 0;
 current_oper := nil;
 currently_operating := false;
 while p <> nil do begin
  case p^.term_type of
   CALC_TERM_TYPE_NUMBER:
```

```
 begin
  if currently_operating then begin
   current_value :=
    current_oper(current_value,
     p^.term_value);
   currently_operating := false;
  end
  else
   current_value := p^.term_value;
 end;
CALC_TERM_TYPE_OPERATOR:
 begin
  current_oper := p^.term_oper_handler;
  currently_operating := true;
 end;
CALC_TERM_TYPE_FUNCTION:
 begin
  if currently_operating then begin
   current_value :=
    current_oper(current_value,
     evaluate_func);
   currently_operating := false;
  end
  else
   current_value := evaluate_func;
 end;
CALC_TERM_TYPE_VARIABLE:
 begin
  if currently_operating then begin
   current_value :=
    current_oper(current_value, x);
   currently_operating := false;
  end
  else
   current_value := x;
 end;
```

```
   CALC_TERM_TYPE_BRACKETOPEN:
    begin
     p := p^.term_next;
     if currently_operating then begin
      current_value :=
       current_oper(current_value,
        get_y_value(x, p));
      currently_operating := false;
     end
     else
      current_value := get_y_value(x, p);
    end;
   CALC_TERM_TYPE_BRACKETCLOSE:
    begin
     get_y_value := current_value;
     exit;
    end;
  end;
  if p <> nil then
   p := p^.term_next;
 end;
 get_y_value := current_value;
end;
{ procedura per il disegno del grafico e
  zoom di esso }
procedure draw_graph;
var x, y : integer;
var vp : viewporttype;
var relmaxx, relmaxy : word;
var barcolor : shortint;
var p : calc_term_t_p;
begin
 getviewsettings(vp);
 with vp do begin
  relmaxx := (x2 - x1) div 2;
  relmaxy := (y2 - y1) div 2;
```

```
  x := ((x2 - x1) div -2);
  repeat
   p := calc_term;
   y := round(get_y_value(x / calc_zoom, p) * calc_zoom);
   if calc_errno = CALC_ERROR_FUNCTIONNOTDEFINED then begin
    barcolor := CALC_GRAPH_PROGRESSBAR_COLOR_NOTDEFINED;
    calc_errno := CALC_NO_ERROR;
   end
   else if (y > relmaxy) or (y < -relmaxy) then
    barcolor := CALC_GRAPH_PROGRESSBAR_COLOR_OOB
   else begin
    putpixel(x + relmaxx, relmaxy - y,
       CALC_GRAPH_LINE_COLOR);
    barcolor := CALC_GRAPH_PROGRESSBAR_COLOR_DEFINED;
   end;
   setcolor(barcolor);
   line(x + relmaxx, (y2 - y1) - CALC_GRAPH_PROGRESSBAR_WIDTH,
    x + relmaxx, (y2 - y1));
   inc(x);
   delay(CALC_GRAPH_DELAY);
  until x > relmaxx;
 end;
end;
{ disegna il piano cartesiano. }
procedure draw_carthesio;
var vp : viewporttype;
var ts : textsettingstype;
var x, y, i : integer;
var relmaxx, relmaxy : word;
var th, tw : integer;
var str : string;
var increment : integer;
begin
 getviewsettings(vp);
 gettextsettings(ts);
```

```
with vp do begin
 relmaxx := x2 - x1;
 relmaxy := y2 - y1;
 moveto(relmaxx div 2, relmaxy);
 lineto(relmaxx div 2, 0);
 moveto(0, relmaxy div 2);
 lineto(relmaxx, relmaxy div 2);
 increment := 40 div calc_zoom;
 if increment = 0 then increment := 1;
 settextstyle(ts.font, horizdir, ts.charsize);
 y := relmaxy div 2;
 x := relmaxx div 2;
 i := increment;
 while x < relmaxx do begin
  x := x + (increment * calc_zoom);
  { stampa il tick. }
  line(x, y + CALC_GRAPH_CARTHESIO_TICK_HALF_LENGTH,
   x, y - CALC_GRAPH_CARTHESIO_TICK_HALF_LENGTH);
  line(relmaxx - x, y + CALC_GRAPH_CARTHESIO_TICK_HALF_LENGTH,
   relmaxx - x,
   y - CALC_GRAPH_CARTHESIO_TICK_HALF_LENGTH);
  { stampa il numero. }
  str := int2str(i);
  th := textheight(str);
  tw := textwidth(str) div 2;
  outtextxy(x - tw, y + th, str);
  outtextxy(relmaxx - x - tw, y + th, '-' + str);
  i := i + increment;
 end;
 settextstyle(ts.font, vertdir, ts.charsize);
 i := increment;
 x := relmaxx div 2;
 while y < relmaxy do begin
  y := y + (increment * calc_zoom);
  { stampa il tick. }
```

```
   line(x - 2, y, x + 2, y);
   line(x - 2, relmaxy - y, x + 2, relmaxy - y);
   { stampa il numero. }
   str := int2str(i);
   th := textheight(str) * 2;
   tw := textwidth(str) div 2;
   outtextxy(x + th, y - tw, '-' + str);
   outtextxy(x + th, relmaxy - y - tw, str);
   i := i + increment;
  end;
  settextstyle(ts.font, ts.direction, ts.charsize);
 end;
end;
{ disegna una barra con del testo all`interno. }
procedure draw_bar(x1, y1, x2, y2 : word; str : string);
begin
 rectangle(x1, y1, x2, y2);
 outtextxy((x2 - textwidth(str)) div 2,
     (y1 + (textheight(str) div 2)), str);
end;
{ procedura principale che disegna il piano,
  disegna la barra di stato, disegna il grafico,
  attende un input dall`utente e rilascia la memoria. }
procedure calc_graph_main;
var bar_width : word;
begin
 setwritemode(copyput); { usa MOV per disegnare i pixel a video }
 bar_width := textheight(calc_func) + CALC_GRAPH_BAR_SHIFT * 2;
 draw_bar(0, 0, maxx, bar_width, { barra superiore }
    'y = ' + calc_func + ' [fattore di zoom: ' +
    int2str(calc_zoom) + 'x]');
 draw_bar(0, (maxy - bar_width), maxx, maxy, { barra inferiore }
    'risoluzione: ' + int2str(maxx + 1) +
    ' x ' + int2str(maxy + 1));
 { setta il viewport al piano cartesiano }
 setviewport(0, bar_width + 1, maxx, maxy - bar_width - 1, clipOn);
```

```
 { disegna gli assi }
 draw_carthesio;
 { disegna il grafico }
 setwritemode(xorput);
 draw_graph;
 { attendi un input dall`utente }
 readkey;
 { libera la memoria }
 cleanup;
end;
{ funzione per verificare la presenza di una modalita`
  grafica utilizzabile, per tornare in modalita` terminale
  e per settare nuovamente la modalita` grafica, utilizzando
  la risoluzione + elevata possibile.
  ritorna false in caso di fallimento }
function calc_setgraph(todo : shortint) : boolean;
var gd, gml, gmh : integer;
begin
 calc_setgraph := false;
 case todo of
  CALC_INIT :
   begin
    registerbgidriver(@EGAVGAdriver);
    registerbgidriver(@CGADriver);
    registerbgidriver(@HERCDriver);
    gd := detect;
    if gd = grOk then begin
     calc_setgraph := true;
     initgraph(gd, gml, '');
     maxx := getmaxx;
     maxy := getmaxy;
    end
    else
     calc_errno := CALC_ERROR_INITGRAPH;
   end;
  CALC_CRT : restorecrtmode;
```

```
  CALC_GRAPH :
   begin
    detectgraph(gd, gml);
    getmoderange(gd, gml, gmh);
    setgraphmode(gmh);
    maxx := getmaxx;
    maxy := getmaxy;
   end;
  CALC_CLOSE : closegraph;
 end;
end;
{ procedura per leggere i dati in input dall`utente }
function calc_read_data : boolean;
var buf : string;
var val_code : integer;
begin
 calc_errno := CALC_NO_ERROR;
 write('Inserire la funzione da disegnare [invio per uscire]: ');
 readln(calc_func);
 if calc_func <> '' then begin
  val_code := 0;
  repeat
   write('Inserire il fattore di zoom [default: ',
    CALC_GRAPH_DEFAULT_ZOOM, ' X]: ');
   readln(buf);
   calc_zoom := CALC_GRAPH_DEFAULT_ZOOM;
   if(buf <> '') then begin
    val(buf, calc_zoom, val_code);
    if val_code <> 0 then
     calc_errno := CALC_ERROR_BADINTEGER
    else begin
     calc_zoom := abs(calc_zoom)
     if calc_zoom > 1000 then
      val_code := 1;
    end;
   end;
```

```
  until val_code = 0;
  calc_read_data := true;
 end
 else
  calc_read_data := false;
end;
{ un simpatico banner :) }
procedure calc_banner;
begin
 writeln('** GRCALC versione ', CALC_VERSION, '.');
 writeln('** Copyright (C) 2002 Marcello Barnaba.');
end;
{$F+}
procedure calc_exitproc;
begin
 writeln;
 writeln('Grazie per aver utilizzato questo programma !');
 write('Premi un tasto qualsiasi per continuare . . .');
 readkey;
end;
{$F-}
{ main. verifica la presenza della modalita` grafica, e se presente,
  inizia un ciclo in cui viene chiesto all`utente la funzione che
  vuol disegnare, viene fatto il controllo della sintassi e viene
  quindi disegnata. per uscire e` sufficiente non immettere alcuna
  funzione. se la modalita` grafica non e` disponibile, il programma
  esce immediatamente. }
begin
 calc_errno := CALC_NO_ERROR;
 exitproc := @calc_exitproc;
 if calc_setgraph(CALC_INIT) then begin
  calc_setgraph(CALC_CRT);
  calc_banner;
  writeln('* Hardware grafico disponibile');
  writeln;
```

```
  while calc_read_data do begin
   if not calc_parse_func(calc_func) then begin
    calc_perror(calc_errno);
    writeln;
   end
   else begin
    calc_setgraph(CALC_GRAPH);
    calc_graph_main;
   end;
   calc_setgraph(CALC_CRT);
  end;
  calc_setgraph(CALC_CLOSE);
 end
 else
  calc_perror(calc_errno);
end.
```