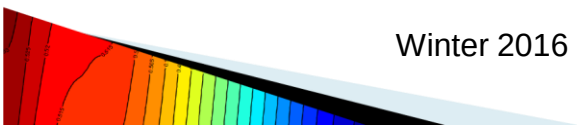
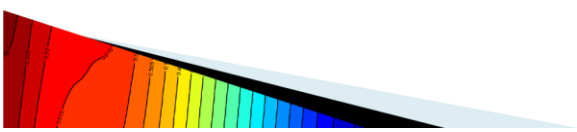
GNG 1106
Fundamentals of Engineering Computation
File IO and Plotting

Gilbert Arbez

Winter 2016



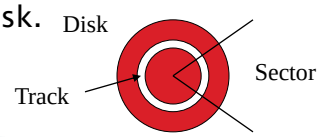
Topic 1: File I/O Library



Files and File Processing

Computing Concepts: Files and Streams

- Files are commonly used for the permanent storage of large quantities of data.
- Files are saved on a secondary storage device such as a hard drive or a floppy disk; these disks can hold files of different types:
  - Text files: `lab1_report.doc`
  - Program files: `lab1.c`
  - Executable files: `lab1.exe`
  - Data files: `cal_quake.dat`
- Files are usually partitioned and written over many tracks and sectors of a disk.

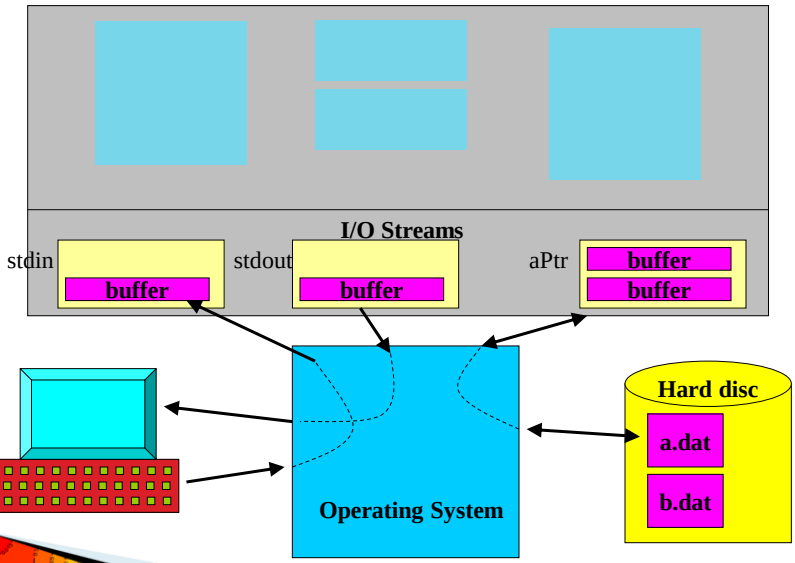


3

- The operating system
  - Maintains a directory tree that lists all files stored on all disks;
  - The directory associates each file to its physical location(s) on the disk; and
  - Maintains the File Control Bloc (FCB) that allows a program to access a file stored on a disk.
- A file can be written or read by a C program.
  - Standard C functions are available and can be used to create and manipulate files.
- A file is viewed in C as a sequence of bytes.
  - When a file is opened, a stream is associated with the file.
  - A stream can be viewed as a communication channel between the program and the opened file.
  - As we have seen, a stream contains memory buffers that are used to exchanged data with the operating system.
  - The stream is a structure manipulated by the standard functions.
  - It also contains a **file position pointer** that maintains a current position in the file where characters or data are read.

4

Programming model for input and output



5

Opening and Closing a File in C

- A file is accessed in C using a pointer to a file I/O stream structure.
  - If more than one file must be opened by the same program, then each file must have its own pointer to a different stream.
- A pointer to a stream structure is declared as follows:

```
FILE *filePtr; /* FILE is a structure type */
```

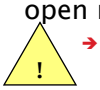
  - The type `FILE` is defined in `stdio.h`
  - The type `FILE` is a structure that has all of the system dependent information required to allow a C program (standard functions) to easily manipulate a file.
  - The type `FILE` is linked to the File Control Bloc.
- A pointer to a file is assigned to a specific file using the standard C function `fopen`:

  E.g.: `filePtr = fopen("test.dat", "w");`

  - `fopen` requires two arguments:
    - the name of the file to be opened, in this example the name is `test.dat`, and
    - the file open mode, in this example the mode is `w`
- A pointer to an I/O stream must be assigned to a file using `fopen` before data can be read from the file or written to the file; `fopen` opens the file and establishes the stream required to access it.

6

- There exists many file open modes.
  - The mode that best suits the type of file access required is selected.
    - `w` write: used to create a new file or to erase an existing file and to write in it.
    - `r` read: used to read an existing file.
    - `a` append: used to write data at the end of a file.
    - `w+` write+: used to create a new file or to erase the content of an existing file, and then to write to and to read from the file.
    - `r+` read+: used to read from and to write to the file.
    - `a+` append+: used to read from a file and to write at the end of the file.
    - `b` binary: added when opening a file as a binary file
- Careful: a particularly nasty error is to open a file using the file open mode `w` for reading.

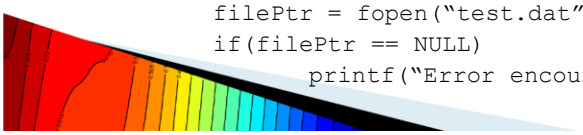


➔ Not only will reading be impossible in this mode but the contents of the file will also be erased!

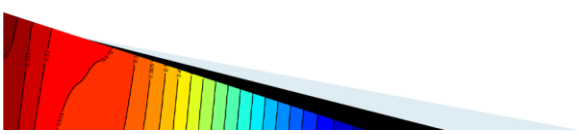
- If there was an error opening the file then `fopen` returns the `NULL` address.

E.g.:

```
FILE *filePtr;
filePtr = fopen("test.dat", "r");
if(filePtr == NULL)
        printf("Error encountered opening file.\n");
```

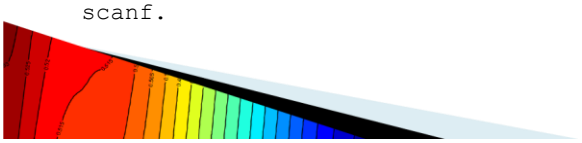


- A file can be closed by invoking the C standard function `fclose`
  - E.g.: `fclose(filePtr);`
  - Normally, a file is closed as soon as we are done accessing it.
  - `fclose` receives one argument: the pointer to the file to be closed.
  - `fclose` is of type `int` (i.e. returns an `int` value).
  - `fclose` returns `0` if it has successfully closed the file, or `EOF` if an error was encountered.
- If files are not closed when execution of `main()` ends then the operating system will close all opened files.
  - It is recommended that all files opened by a program be closed as soon as they are no longer needed since this releases the resources required to manage open files.
- The prototypes of `fopen` and `fclose` are found in `stdio.h`
  - This header file must therefore be included before `main()` if `fopen` and `fclose` are required.



Sequential ASCII Files: Reading and Writing

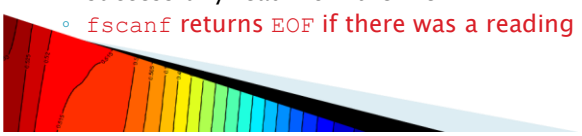
- An ASCII file is simply a file consisting of a sequence of bytes containing bit patterns for ASCII characters.
- The advantage of an ASCII file is that it can be read or written using a text editor (Notepad).
- The disadvantage of an ASCII file is that it can become quite large.
  - The number `-32768` for example, when stored in a variable of type `int`, requires four bytes of storage, whereas that same number stored in an ASCII file, requires six bytes.
- Many C standard functions are available for reading and writing ASCII files.
  - `fprintf` similar to `printf`.
  - `fscanf` similar to `scanf`.
  - `fgetc` reads a character.
  - `fputc` writes a character.
  - `fgets` reads a string.
  - `fputs` writes a string.
  - The prototype for these functions are found in `stdio.h`
  - `fprintf` and `fscanf` use the same conversion specifiers as `printf` and `scanf`.



- A message string and/or variables can be written to a open file using `fprintf`
  - The usage of `fprintf` is similar to the usage of `printf`.
  - The first argument in `fprintf` is the pointer to the open file where the data is to be written.

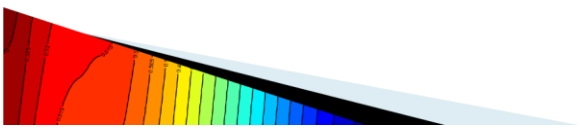
    E.g.: `fprintf(filePtr, "%d %f\n", integer_1, real_1);`
  - The sequential execution of many `fprintf`'s will creates the content of a sequential file.
  - `fprintf` returns an `int` corresponding to the number of characters successfully written to the file.
  - `fprintf` returns a negative number if a writing error was encountered.
- Data can be read from an open file using `fscanf`
  - The usage of `fscanf` is similar to the usage of `scanf`.
  - The first argument in `fscanf` is the pointer to the open file to be read.

    E.g.: `fscanf(filePtr, "%d%f", &integer_1, &real_1);`
  - The above will read a line written by the previous call to `fprintf`
  - The sequential execution of many `fscanf`'s will read a sequential file.
  - `fscanf` returns an `int` corresponding to the number of items successfully read from the file.
  - `fscanf` returns `EOF` if there was a reading error.



- The function `fgets` is useful for reading a string of characters from an open file.

  E.g.:
  ```
  FILE *filePtr;
  char string[10];
        ⋮
  fgets(string, 10, filePtr);
  ```
  - In the above example, `fgets` reads in at most `10-1=9` characters from the open file pointed to by `filePtr` and stores these characters into the array `string`.
  - In this case, reading stops when `9` characters are read, or when a new line character (ASCII 10) is read or the end of file is found.
    - Note that the space, if found in the file, is not treated as string delimiters by `fgets` whereas `scanf` and `fscanf` interpret them as delimiters.
  - If read, the new line character is retained in `string`.
  - The null character `'\0'` is stored in the string immediately after the last character read in from the file.
  - If an error was encountered during reading then `fgets` returns the `NULL` address.
- The standard C function `gets` exists for reading in strings from the keyboard but it does not work in the same way as `fgets`.
  - In `gets` the caller cannot specify the maximum number of characters to be read from the input stream.
  - It is preferable to use "`fgets(string, 10, stdin)`" to read from the keyboard.

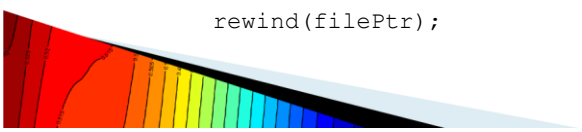


11

- The function `fgetc` is useful for reading a file one character at a time (recall `getchar`).

  E.g.:
  ```
  FILE *filePtr;
  int character;
    ⋮
  character = fgetc(filePtr);
  ```
  - `fgetc` returns the next character in the file pointed to by `filePtr`
  - The character read is returned as an `int`.
  - If a reading error was encountered then `fgetc` returns `EOF`.
- It may be desirable to read many times the same file during a single execution.
  - To re-read a file from the beginning, we must first ensure that the file position pointer is re-positioned at the beginning of the file.
  - The file position pointer points to the location or byte in the file where the next read or write operation will occur.
  - The C standard function `rewind` (in `stdio.h`) can be used to re-position the file position pointer to the top of the file.

  E.g.:
  ```
  FILE *filePtr;
    ⋮
  rewind(filePtr);
  ```



12

Detecting the End of a File

- A mechanism for detecting the end of a file when reading is obviously required.
- There exists three mechanisms in C for detecting the end of a file.
  - Reading the file in a `for` loop if we know ahead of time the number of lines to be read.
    - E.g.: The file `Conv_DR1.dat` has 37 lines of numerical data to read.
    - E.g.: By convention, the first line of a file is often the number of lines to read.
  - Reading the file in a sentinel controlled `while` loop if we know ahead of time that a sentinel value will be placed on the last line of the file.

E.g.:
```
100.0
1000.0
1500.0
  ⋮
-99.0
```

File containing the altitude above sea level of a weather balloon over time. The altitude must always be a positive number (almost always…) so a negative value such as `-99.0` could be used to indicate the last line in the file.

  - If we do not know ahead of time the number of lines in the file or if there is no sentinel value in the file to indicate the last line, then we can use a `while` loop and detect the end of the file by:
    - testing that the function used to read the file has not encountered a reading error, or
    - using the function `feof` to determine if we are at the end of the file.

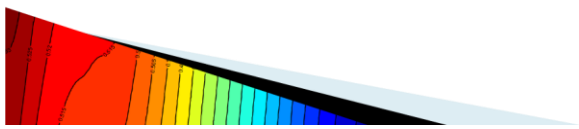
- A reading error will occur if the caller attempts to read past the last character in the file.
  - Recall:
    - `fscanf` returns `EOF` if a reading error occurred.
    - `fgetc` returns `EOF` if a reading error has occurred.
    - `fgets` returns the `NULL` address if a reading error has occurred.
- The function `feof` detects the end of a file.

E.g.:
```
FILE *filePtr;
⋮
if( feof(filePtr) )
printf("End of file detected.\n");
```

  - The function `feof` requires the file pointer as the argument and returns a non-zero integer if and only if the file position pointer is at the end of the file.
    - ➔ `feof` **returns true** if and only if the file position pointer is at the end of the file.
  - The prototype for the function `feof` is located in `stdio.h`

Sequential Binary Files: Reading and Writing

- A binary file is simply a sequence of bytes that have been copied directly from memory (RAM) into a file; a binary file is in effect a memory dump.
- The advantages of a binary file are listed below.
  - Binary files are usually smaller compared to an ASCII file containing the same information.
  - Input and output operations between a binary file and a program are usually faster than input and output operations with an ASCII file since data conversions are not required.
    - E.g.: A real number, read in as a sequence of bytes in an ASCII file (where each byte represents a digit or the decimal point), must be converted using `%f` to a `double` before it is stored in a variable of type `double`. Such a conversion is not necessary when reading a binary file since the binary representation of a real number for example, is read and stored directly into a `double` variable.
- The disadvantages of a binary file are listed below.
  - In general they cannot be written or read using a text editor.
  - They are severely limited in portability since they must be read using variables that have the same binary representation as the stored data.
    - ➔ Limited portability of data between programming languages and machines.



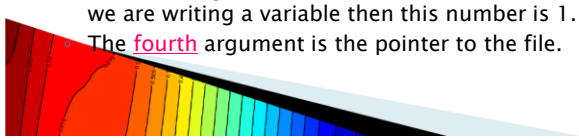
- Binary files are opened and closed just like ASCII files using `fopen` and `fclose`.
- The C standard functions `fwrite` and `fread` can be used to write and read binary files.
  - The prototypes are found in `stdio.h`
- `fwrite` can be used to write a prescribed number of bytes to a file starting from a specific memory address.

E.g.:

```
FILE *filePtr;
int integer = 100;
double real = 7.5;
double dbl[3] = {1.0, 1.1, 1.2};
filePtr = fopen("test.bin", "wb");
fwrite(&integer, sizeof(int), 1, filePtr);
fwrite(&real, sizeof(double), 1, filePtr);
fwrite(dbl, sizeof(double), 3, filePtr);
```

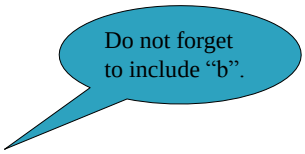
Do not forget to include "b".

  - The first argument in `fwrite` is the address of the first byte to be stored in the file.
  - The second argument is the number of bytes to write.
  - The third argument corresponds to the number of elements of an array to write; if we are writing a variable then this number is 1.
  - The fourth argument is the pointer to the file.

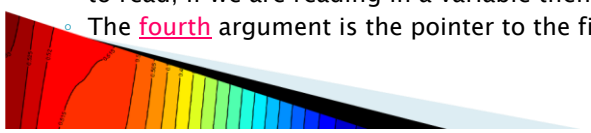


- `fread` can be used to read a prescribed number of bytes from a binary file into memory starting from a specific memory address.
  - The file to be read must be opened in file open mode `r` or `w+` or `r+`
    E.g.: Assuming that we want to read in binary data from the file `test.bin` created on the previous page:

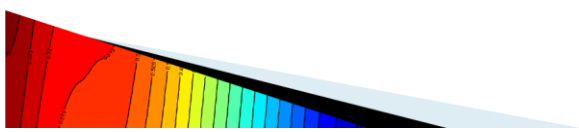
```
FILE *filePtr;
int integer;
double real;
double dbl[3];
filePtr = fopen("test.bin", "rb");
fread(&integer, sizeof(int), 1, filePtr);
fread(&real, sizeof(double), 1, filePtr);
fread(dbl, sizeof(double), 3, filePtr);
```



  - The first argument in `fread` is the address of the first byte in memory where the data read in from the file will start being stored.
  - The second argument is the number of bytes to read.
  - The third argument corresponds to the number of elements of an array to read; if we are reading in a variable then this number is 1.
  - The fourth argument is the pointer to the file.

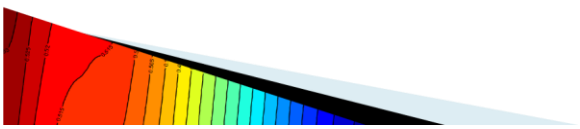


- The function `fwrite` returns the number of items successfully written to the file.
  - If a writing error occurred, the number returned by `fwrite` will be smaller than the value of the third argument in its argument list.
- The function `fread` returns the number of items successfully read from the file.
  - If a reading error occurred or if the end of file was detected then the returned value will be smaller than the value of the third argument in its argument list.

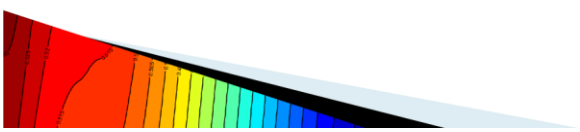


Examples

- Using text (ASCII) files
  - asciiFile.c
  - A simple program that creates an ASCII file and reads from it.
- Using binary files
  - binaryFile.c
  - Writes the contents of a structure into a binary file
  - Reads from a binary file the contents of a structure

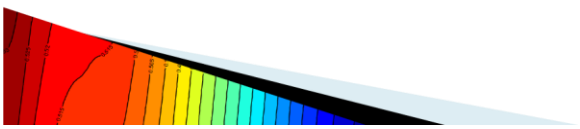


Topic 2: PLplot Library



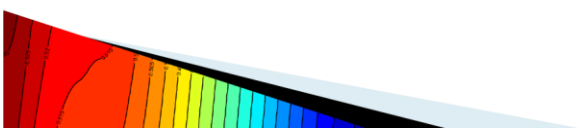
Library for Plotting Graphs

- Solving engineering problems requires a capability to plot data.
  - As engineers and scientists viewing data on plots makes analysis much easier.
- PLplot:
  - Is a library of C functions for making scientific plots; it is available for a number of programming languages, including
    - C/C++, Fortran95, Java, Octave, Perl, Python, and Tcl/Tk.
  - Can create standard x-y plots, semi-log plots, log-log plots, contour plots, 3D plots, shade (gray-scale, and color) plots, mesh plots, bar charts and pie charts.
  - Multiple graphs (same or different sizes) can be place on a single page with multiple curves in each graph.
  - PLplot was developed in the 1980's on VAX mainframe computers and ported to the PC. Its latest version is 5.11.1 and runs in an number of environments as mentioned above.



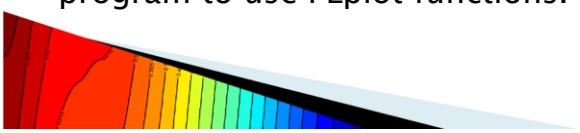
Using PLplot in GNG1106

- The following slides introduce the main functions of the library to create plots for this course using CodeBlocks.
  - More detailed documentation is provided in the PDF file posted in the Lab page of the course Web site. The Programming Section will be of interest.
- The library has been compiled for use with CodeBlocks
  - The library is available on the computers in the SITE labs.
  - The library is also included in the installation file provided on the Course Web Site.
  - To have access to the PLplot library, use the C Console App/PLplot Project in CodeBlocks.
  - These projects are configured to include the PLplot library and include a template which is a program that plots a sine function.



Creating a Plot using PLplot

- To draw a graph, at least 4 PLplot functions need to be called:
  - `plinit` to initialize the plot.
  - `plenv` to define the range and scale of the graph, axes, etc. (`pllab` can be used to set the labels on the graph).
  - One or more calls (can draw more than one curve on the set of axes) to `plline` or `plpoin` to draw lines or points.
  - `plend`, to close the plot.
- PLplot has defined a special types for the function parameters.
  - `PLINT` (defined as a `long`, this is an integer type).
  - `PLFLT` (define as `double`).
- Include the header file "gng1106plplot.h" in your program to use PLplot functions.

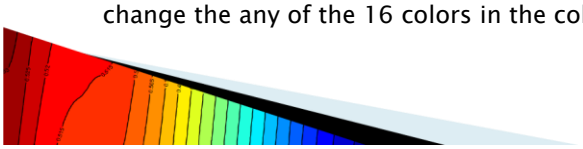


Initialising the Plot

- Library Function: `void plinit(void)`
  - Sets up all internal data structures for plotting and initializes the output device.
  - A number of function can be called to configure plotting (call before calling `plinit`).

    `void plsdev(char *devname)` to define the plotting device.
    - Use "wingcc" as the device name with CodeBlocks, that is use the following call to setup the plot device.

      ```
      plsdev("wingcc");
      ```
    - `void plssub(PLINT nx, PLINT ny)` to create subplots (see slide that describes `plssub`)
    - `void plscolbg(PLINT r, PLINT g, PLINT b)` – to change the background color
      - The background color default is white.
    - `void plscol0(PLINT icol0, PLINT r, PLINT g, PLINT b)` – to change the any of the 16 colors in the color pallet.



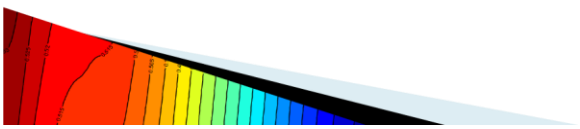
Setting the Graph Scale

- Library Function

```
void plenv(PLFLT xmin, PLFLT xmax, PLFLT ymin, PLFLT ymax,
           PLINT just, PLINT axis)
```

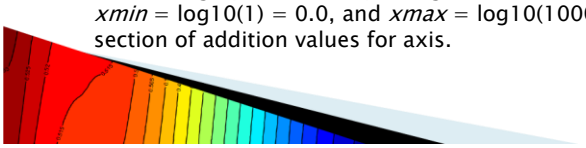
- Setups the scale and appearance of the axes. Note that `plenv` calls `pladv` to advance to next subplot.

- Parameters:
  - xmin, xmax: Sets the left and right limits of the horizontal scale.
  - ymin, ymax: Sets the left and right limits of the vertical scale.
  - just: Zero or One. Zero, each axis is scaled independently, One, both axis have the same scale. Use zero.
  - axis: configures the axes with box, tick marks, labels, and/or grid as follows:



Setting the Graph Scale (continued)

- The axis parameter can be set to one of the following values to setup the axes scales.
  - -2: No box or annotation.
  - -1: Draw box only.
  - 0: Draw box, labelled with coordinate values around edge.
  - 1: In addition to box and labels, draw the two axes X = 0 and Y = 0.
  - 2: Same as *axis* = 1, but also draw a grid at the major tick interval.
  - 10: Logarithmic X axis, linear Y axis.
  - 11: Logarithmic X axis, linear Y axis and draw line Y = 0.
  - 20: Linear X axis, logarithmic Y axis.
  - 21: Linear X axis, logarithmic Y axis and draw line X = 0.
  - 30: Logarithmic X and Y axes.
- Logarithmic axes only affect the appearance of the axes and their labels
  - It is up to the user to compute the logarithms prior to passing them to plenv and any of the other routines.
  - Thus, if a graph has a 3-cycle logarithmic axis from 1 to 1000, we need to set *xmin* = log10(1) = 0.0, and *xmax* = log10(1000) = 3.0. See plenv in reference section of addition values for axis.

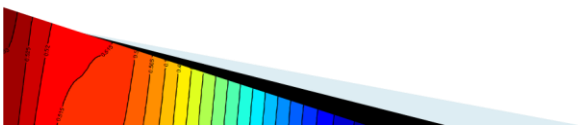


Labelling the Graph

- Library Function

```
void pllab( char *xlabel, char *ylabel,
            char *tlabel)
```

- Adds labels to axes and title of the graph.
- Parameters
  - `xlabel`: label for the horizontal axis
  - `ylabel`: label for the vertical axis
  - `tlabel`: label for the graph title.



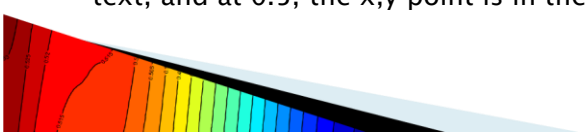
27

Labelling the Graph

- Library Function

```
void plptex(PLFLT x, PLFLT y, PLFLT dx, PLFLT dy,
            PLFLT just, char *string)
```

- Adds a string anywhere on the plot. The reference line passes through the middle of the text from the right edge.
- Parameters
  - `x, y`: Coordinates of a reference point for positioning the reference line (see `just`).
  - `dx, dy`: specifies the angle of the text, i.e. the reference line passes through the points (x,y) and (x+dx, y+dy).
  - `just`: Justifies the text relative to the x, y point. The value of just specifies where the reference point is located on the reference line. If 0.0, the x.y point is at the left-hand edge of the text, if at 1.0 the x,y point is at the right-hand edge of the text, and at 0.5, the x,y point is in the middle of the text.



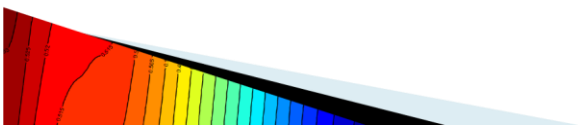
28

Drawing Lines (curves)

- Library Function

```
void plline(PLINT n, PLFLT *x, PLFLT *y)
```

- Draws a curve on the graph. All data for drawing the curve are provided in the two referenced arrays (arrays contains **double** values).
- Parameters
  - **n**: number of points in the arrays
  - **x**: pointer to array of x coordinate
  - **y**: pointer to array of y coordinate.

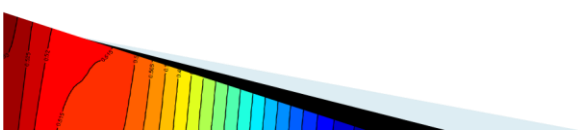


Drawing Points

- Library Function

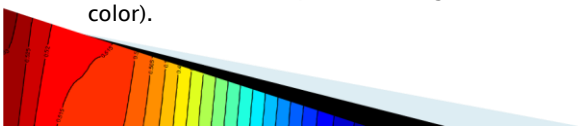
```
void plstring(PLINT n, PLFLT *x, PLFLT *y, char *string)
```

- Adds a character at each of the n points defined by the arrays x and y. Although the PLplot documentation describes the functions **plpoin** and **plsym** in the chapter "Simple Use of PLplot", its documentation of these functions indicates that **plstring** has largely superseded these functions. The function **plstring** gives access to any character and is simpler to use.
- Parameters
  - **n**: number of points in the arrays
  - **x**: pointer to array of x coordinate
  - **y**: pointer to array of y coordinate.
  - **string**: A character to plot. Although it is possible to include more than one character in the string, it is rarely practical to do so.



Using Colors

- PLplot uses a color map to allow selection of colors. Colors are defined in terms of mixing red, green and blue (represent by the values r, g, b). Values for r, g, and b can range from 0 to 255. Thus the following combinations gives basic colors
  - 0, 0, 0 gives black (no color)
  - 255, 255, 255 gives white (maximum of all ,
  - 255, 0, 0 gives red
  - 0, 255, 0 gives green
  - 0, 0, 255 gives blue
- The PLplot color map can be set during initialisation of the graph. By default, white is the background color and black the foreground color. The library has been configured to use these default colors since they are more practical to include in documents.
  - The color map is simply an array with 16 elements which defines the colors. The index into the array represents the color.
  - Functions (see the next slide) can be used to change the color in the map (i.e. set the value of the array element to get the desired color or to select the desired color).



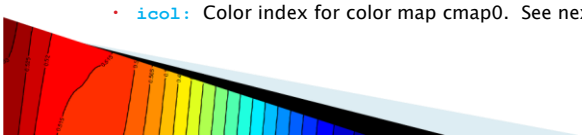
Color Functions

- Library Function

  `void plscolbg(PLINT r, PLINT g, PLINT b)`
  - Sets the background color. Configures the color 0 (index 0 in the PLplot color map 0, cmap0). Must be called before `plinit()`.
  - Parameters
    - `r, g, b`: Values with range 0 to 255 that defines the amount of red, green, and blue included in the color.
- Library Function

  `void plscol(PLINT icol, PLINT r, PLINT g, PLINT b)`
  - Sets the any color in the color map. Configures the color icol (index icol in the PLplot color map 0, cmap0). Must be called before `plinit()`.
  - Parameters
    - `icol:` Color index for color map cmap0. Can be set to values between 0 to 15 (0 is default background color, and 1 is default foreground color).
    - `r, g, b`: Values with range 0 to 255 that defines the amount of red, green, and blue included in the color.
- Library Function

  `void plcol(PLINT icol)`
  - Selects the color for drawing. Can be used to change the foreground color anytime after `plinit()` is called.
  - Parameters
    - `icol:` Color index for color map cmap0. See next slide

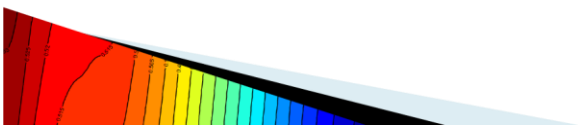


Color Map

- In this course, you will be working with color map 0 (cmap0). Color map 1 is suited to creating shading and useful for 3D plots.
- The header file "*gng1106plplot.h*" contains symbolic color definitions which makes selection of colors much easier and clearer in code.
- The color symbolic constants defined in *gng1106plplot.h* are:

WHITE	0	BROWN	8
BLACK	1	BLUE	9
YELLOW	2	BLUEVIOLET	10
GREEN	3	CYAN	11
AQUAMARINE	4	TURQUOISE	12
PINK	5	MAGENTA	13
WHEAT	6	SALMON	14
GREY	7	RED	15

- The symbolic constants WHITE, BLACK, … can be used as the "icol" arguments values in the color functions. E.g. the call `plcol(RED);` sets the foreground color i.e. drawing color to red.



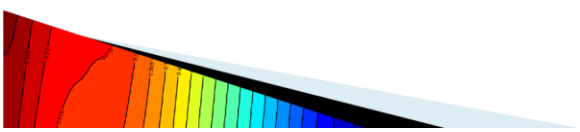
33

Pen Width

- Library Function

```
void plwidth(PLFLT width)
```

- Sets the pen width.
- Parameters
  - `width`: If zero, is interpreted as the minimum pen width the device is printed. If negative, no change occurs. The interpretation of positive values for the pen width is device dependent.



34

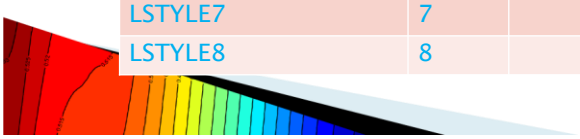
Line Style

- Library Function

```
void pllsty(PLINT lin)
```

- Changes the style of the line drawn (e.g. to dashed lines).
- Parameters
  - `lin`: Its value between 1 and 8 sets the line style as follows. Symbolic constant definitions are provided in gng1106plplot.h header file.

Symbolic Constant	Value	Description
SOLID	1	Continuous line
SHRTDASH_SHRTGAP	2	Short dashes and gaps
LNGDASH_LNGGAP	3	Long dashes and gaps
LNGDASH_SHRTGAP	4	Long dashes with short gaps
LNGDASH_SHRTDASH	5	Long dash – short dash
LSTYLE6	6	
LSTYLE7	7	
LSTYLE8	8	



35

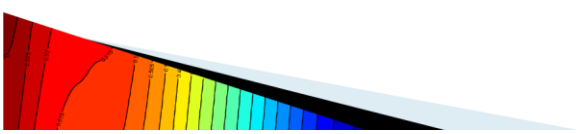
Ending the Plot

- Library Function

```
void plend()
```

- Ends plotting. Changes can be made to the plot until this function is called. In Windows this function returns after the PLplot window is closed.

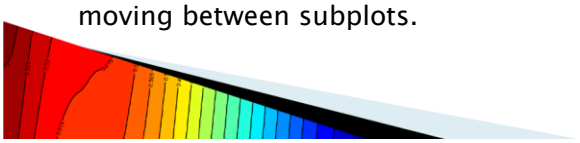
- Other useful functions for controlling the plot
  - void plclear() : clears the plot to start over.
  - void pllegend(): for annotating curves in a plot.
  - See the PLplot documentation for more details.



36

Using Subplots

- PLplot uses the notion of the page. By default a single plot is created on a page after `plinit` is called.
  - PLplot provides functions to divide the page into subplots which allows the creation of multiple plots.
  - It is also possible to create multiple pages, but this is beyond the scope of this course.
- Each subplot can created using the functions already presented
  - The function `pladv` is used to set the focus on the subplot for drawing curves.
  - `plenv` calls `pladv(0)` before configuring the plot environment, (i.e move to the next subplot).
  - Use all drawing functions already presented after moving the desired subplot.
  - See the subsequent slides for details on creating subplots and moving between subplots.

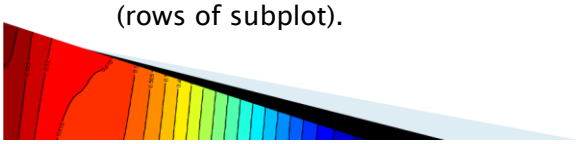


Creating Subplots

- Library Function

  ```
  void plstar(PLINT nx, PLINT ny)
  ```

- Creates a page with nx by ny sub plots.
  - The subplots are numbered from 1 to N (where N is nx X ny).
  - Subplots number from 1, top left, and increase along rows.
    - E.g. 2 x 2 subplots: 1 top-left, 2 top right, 3 bottom-left, 4 bottom-right).
  - This function calls `plinit()` and `plsub()` (`plsub()` is the function that creates the subplots). Thus use this function instead of `plinit()`.
- Parameters
  - `nx, ny`: defines the number of subplots as a matrix where `nx` is the number of subplots in the x direction (column of subplots) and `ny` the number of subplots in the y direction (rows of subplot).



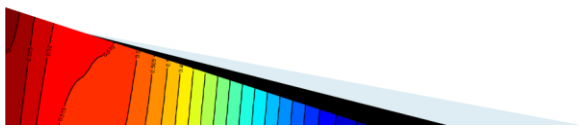
Creating subplots

- Library Function

```
void plssub(PLINT nx, PLINT ny)
```

- Creates a number of subplots in the current page. Should be called after calling `plinit()`. The function `plstar()` calls both `plinit()` and `plssub()`.

- Parameters
  - `nx, ny`: defines the number of subplots as a matrix where `nx` is the number of subplots in the x direction (column of subplots) and `ny` the number of subplots in the y direction (rows of subplot).



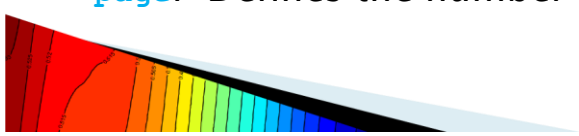
39

Moving between subplots

- Library Function

```
void pladv(PLINT page)
```

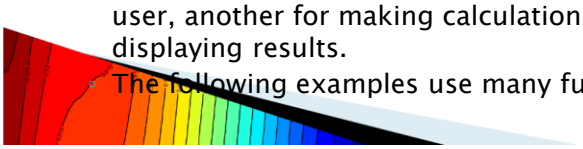
- Moves to a subpage.
  - If `page` is 0, then move to the next subplot.
  - If a positive number, then move to the specific subplot (see `plstar` or `plssub` for details on subplot numbering).
  - The function `plenv` calls this function to advance to the first subplot use `plenv0` if you do not want to advance when setting up the plot.
- Parameters
  - `page`: Defines the number of the subplot.



40

On Using Functions

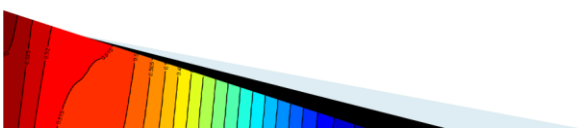
- Up to this point, we have been using a program with two functions (a few may have 3 functions)
  - **`main`**:
    - This function is required, execution starts with this function.
    - We have typically used main to interact with the user.
  - Second function
    - A second function is used typically to make some calculations or process data (e.g. scanning an array).
- Any number of functions can be used in a program
  - The main function should be kept simple and call other functions.
  - For e.g., can have one function get input data from the user, another for making calculations, and a third for displaying results.
  - The following examples use many functions.



41

Example: Plotting a sine wave

- The project PlotSine (with program plotSine.c) plots a sine wave.
  - It is actually the template that is provided when creating a C Console App/PLplot CodeBlocks project.



42

Next Module