PTLIB DEVELOPER MANUAL

When including the ptlib.h file, you’ll get access to simple functions for the command line that abstract away boilerplate code in C. Those functions serve two purposes:

1) get raw input from the command line
2) draw simple graphics to the command line

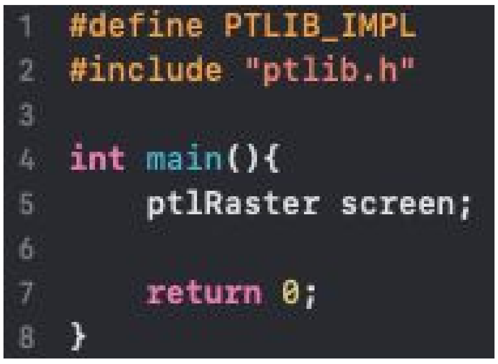
The goal of ptlib is to be simple and easy to use. For this reason only the bare minimum of input and graphics functions are provided. In order to include the library you write:

```
#define PTLIB_IMPL
#include "ptlib.h"
```

at the top of your source file.

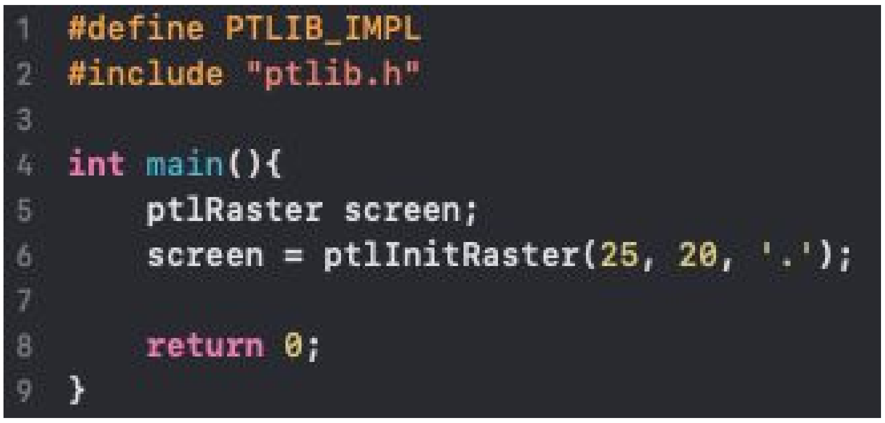
the raster:

The central object that both graphics and input will refer to, is the ptlRaster.
The raster can be thought of as the “screen”, that will also hold the pressed keys.



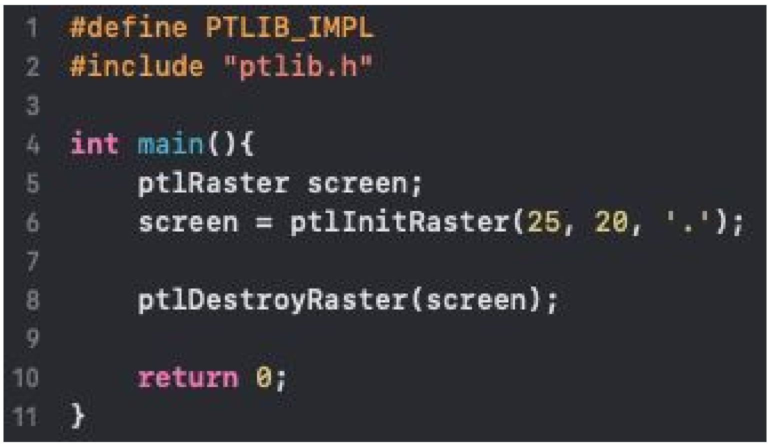
```c
#define PTLIB_IMPL
#include "ptlib.h"

int main(){
    ptlRaster screen;

    return 0;
}
```

after declaring a variable of type ptlRaster, you want to use the function ptlInitRaster() to initialize the raster, with this initialization you’ll have to specify the width of the raster, the height and the character you want to use for the background.



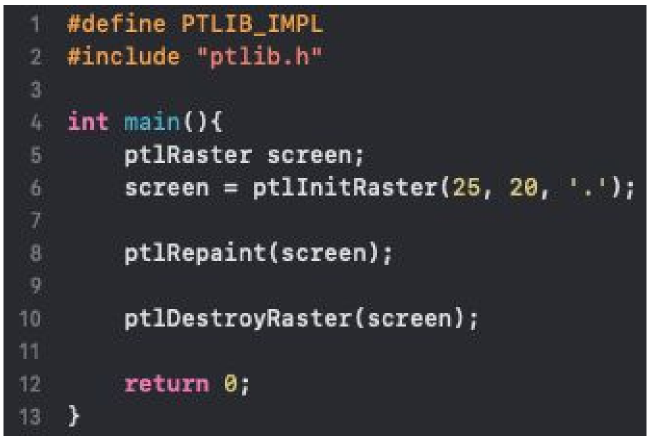
```c
#define PTLIB_IMPL
#include "ptlib.h"

int main(){
    ptlRaster screen;
    screen = ptlInitRaster(25, 20, '.');

    return 0;
}
```

Once you've initialized your raster, it is **VERY** important to also add the function ptlDestroyRaster() in your code. If you don't, then the terminal window input settings will be messed up after the program has finished.



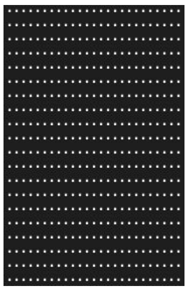
```c
#define PTLIB_IMPL
#include "ptlib.h"

int main(){
    ptlRaster screen;
    screen = ptlInitRaster(25, 20, '.');

    ptlDestroyRaster(screen);

    return 0;
}
```

This won't draw anything to the screen just yet though. What we've done so far is create and initialize a raster object that holds all kinds of information, such as the width, height and background. After we've initialized that raster we make sure to add the ptlDestroyRaster() so the terminal window won't be messed up after our program has finished.
In order for us to see this raster in our command line, we need to call the function ptlRepaint(), which takes a ptlRaster as a parameter. This will clear the screen and draw the raster.



```c
#define PTLIB_IMPL
#include "ptlib.h"

int main(){
    ptlRaster screen;
    screen = ptlInitRaster(25, 20, '.');

    ptlRepaint(screen);

    ptlDestroyRaster(screen);

    return 0;
}
```

If we run that, we'll see nothing. That's because when we call ptlDestroyRaster it moves the raster out of our sight. However, when we scroll up we'll see the raster with a width of 25 characters (or pixels) and a height of 20 characters. Those characters are the '.' character. This is the result:

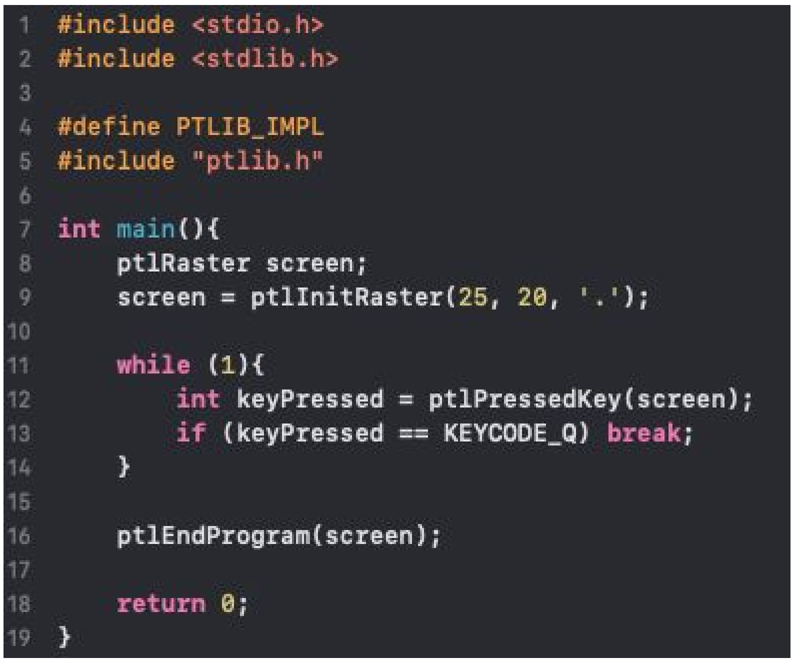


```
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
```

Now we know how to initialize and draw a ptlRaster, we'll first take a look at input, because that's the simplest, afterwards we'll talk about the specific graphics drawing functions.

raw input:

Let's start off with the by far simplest part of the library, raw input.
there is only one function the library provides, called ptlPressedKey(). This function takes a ptlRaster as its parameter, because the raster holds not only information about the graphics, but also about the inputs. The function ptlPressedKey() returns an integer, which we can compare to enum elements the library provides, those being "KEYCODE_" + "character".
Here is an example of a simple program that doesn't draw anything but just waits until the user presses "q", then it quits:



```c
#include <stdio.h>
#include <stdlib.h>

#define PTLIB_IMPL
#include "ptlib.h"

int main(){
    ptlRaster screen;
    screen = ptlInitRaster(25, 20, '.');

    while (1){
        int keyPressed = ptlPressedKey(screen);
        if (keyPressed == KEYCODE_Q) break;
    }

    ptlEndProgram(screen);

    return 0;
}
```

You should call ptlPressedKey() only **ONCE** each iteration and store that return value in an integer. If you check the input multiple times during an iteration, it's possible that you'll end up with different inputs in the same iteration. This is bad and you should avoid it at all costs. The correct way of checking inputs with this library is to create an integer like "keyPressed", and then at the very start of each iteration call ptlPressedKey() **ONCE** and store the return value in the integer "keyPressed". then you never call ptlPressedKey() in the scope of that loop again, but you always work with the integer "keyPressed". You can see a correct example of handling inputs in the image above.

The return values can be the following integers:

- "KEYCODE_" + 'character'   (e.g. KEYCODE_A, KEYCODE_B, …)
- "KEYCODE_" + number   (e.g. KEYCODE_0, KEYCODE_5, …)
- "KEYCODE_" + arrow   (e.g. KEYCODE_UP_ARROW, …)
- and there's 3 special keycodes:
  KEYCODE_SPACE, KEYCODE_TAB, KEYCODE_ENTER

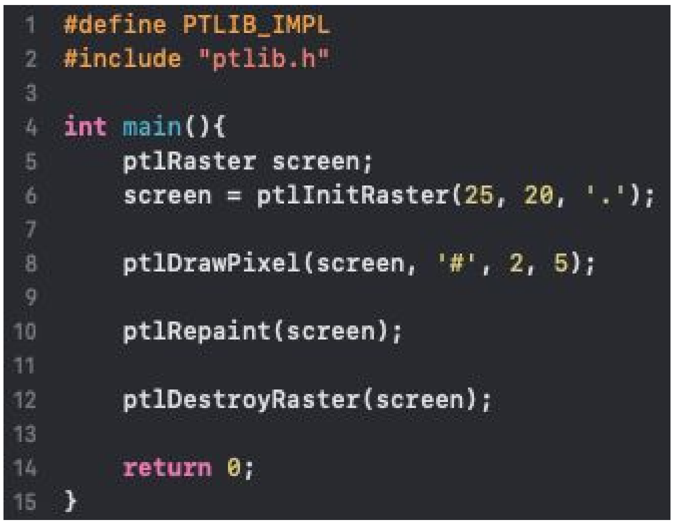
graphics:

every graphics drawing function in the ptlib will take a raster as a parameter, and change some of the contents in that raster. After those contents are changed you can call ptlRepaint() again to see the changes. It is important to know that the pixel in the top left corner is at coordinate (0, 0).
Here’s a quick overview of the graphics drawing functions:

1 - ptlDrawPixel():

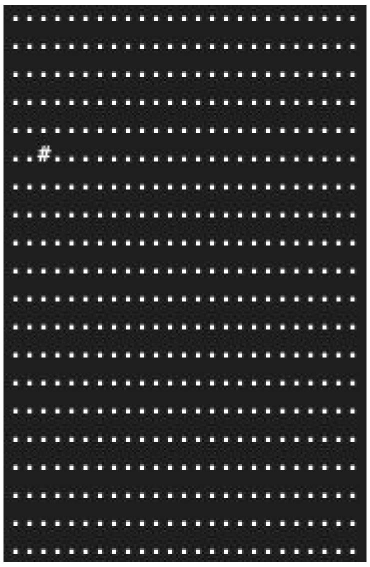
ptlDrawPixel does what it looks like: it draws a pixel in a raster. It takes a raster to draw the pixel in, a char to draw at that coordinate, a value on the x axis and a value on the y axis.

code:



```c
#define PTLIB_IMPL
#include "ptlib.h"

int main(){
    ptlRaster screen;
    screen = ptlInitRaster(25, 20, '.');

    ptlDrawPixel(screen, '#', 2, 5);

    ptlRepaint(screen);

    ptlDestroyRaster(screen);

    return 0;
}
```

result :



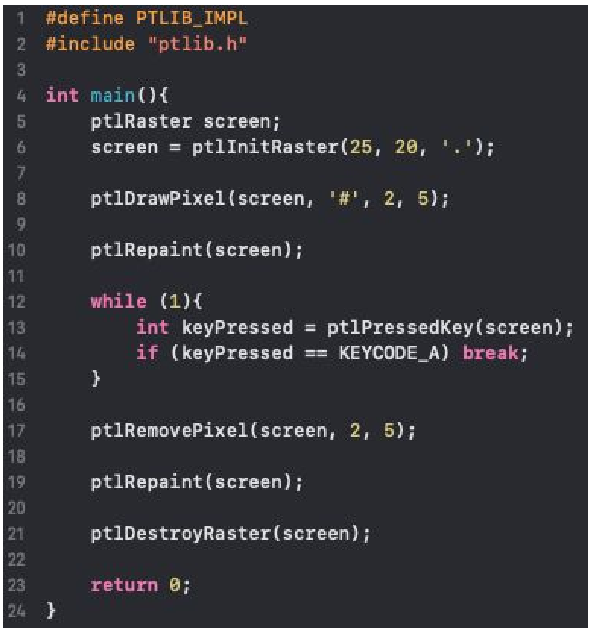
```
.........................
.........................
.........................
.........................
.........................
..#......................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
```

(if you’re running this code yourself, then you’ll have to scroll up in the terminal to see the raster)

2 - ptlRemovePixel():

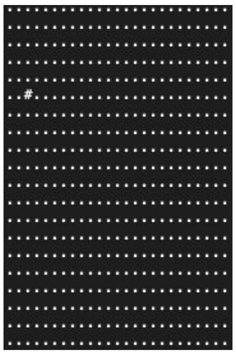
ptlRemovePixel will remove a pixel (or character) at a certain coordinate, and change the character at that coordinate to the background character.

code:



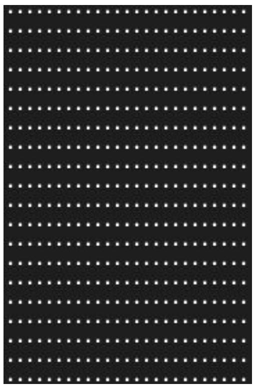
```c
#define PTLIB_IMPL
#include "ptlib.h"

int main(){
    ptlRaster screen;
    screen = ptlInitRaster(25, 20, '.');

    ptlDrawPixel(screen, '#', 2, 5);

    ptlRepaint(screen);

    while (1){
        int keyPressed = ptlPressedKey(screen);
        if (keyPressed == KEYCODE_A) break;
    }

    ptlRemovePixel(screen, 2, 5);

    ptlRepaint(screen);

    ptlDestroyRaster(screen);

    return 0;
}
```

result (before pressing “a”):



```
.........................
.........................
.........................
.........................
.........................
..#......................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
```

result (after pressing “a”):

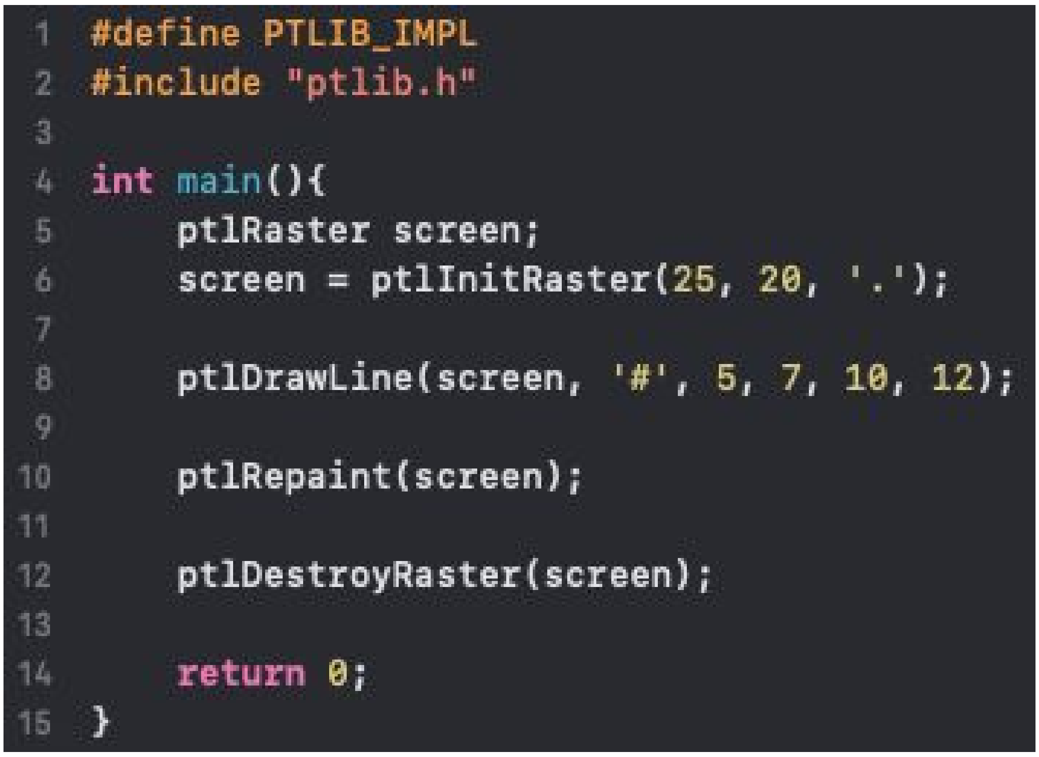


```
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
```

3 - ptlDrawLine():

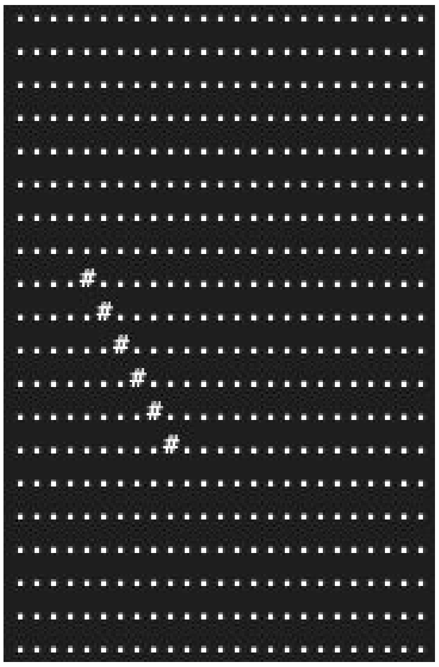
ptlDrawLine will draw a line from one point to another, it takes quite a lot of parameters: ptlRaster, pixelChar(the char of which the line will be made), start_x, start_y, end_x and end_y. Here’s what it looks like:

code:



```c
#define PTLIB_IMPL
#include "ptlib.h"

int main(){
    ptlRaster screen;
    screen = ptlInitRaster(25, 20, '.');

    ptlDrawLine(screen, '#', 5, 7, 10, 12);

    ptlRepaint(screen);

    ptlDestroyRaster(screen);

    return 0;
}
```

result:

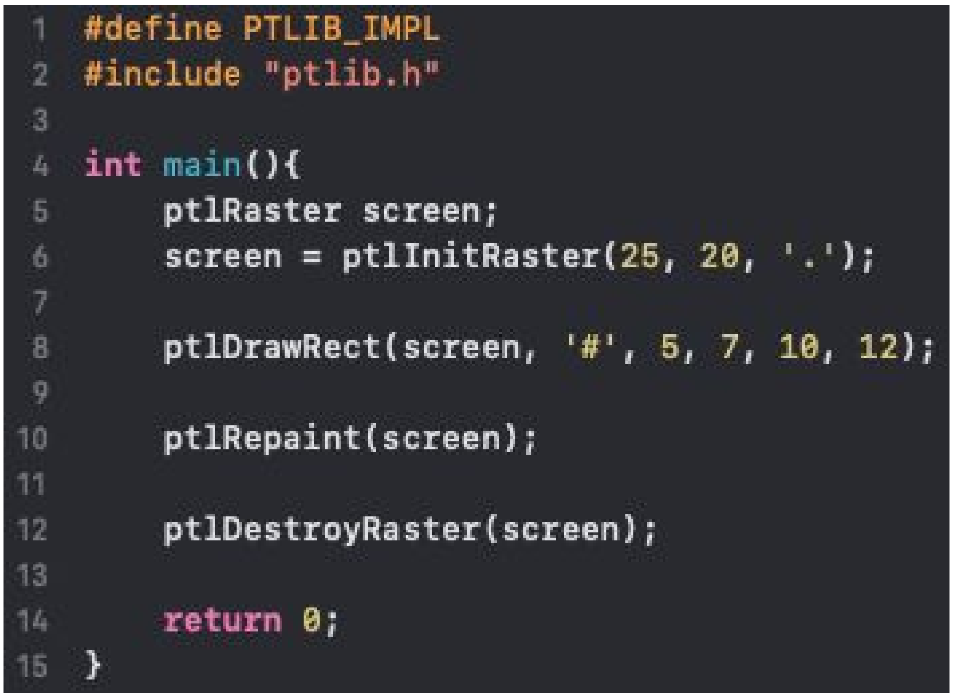


```
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
....#....................
.....#...................
......#..................
.......#.................
........#................
.........#...............
.........................
.........................
.........................
.........................
.........................
.........................
```

4 - ptlDrawRect():

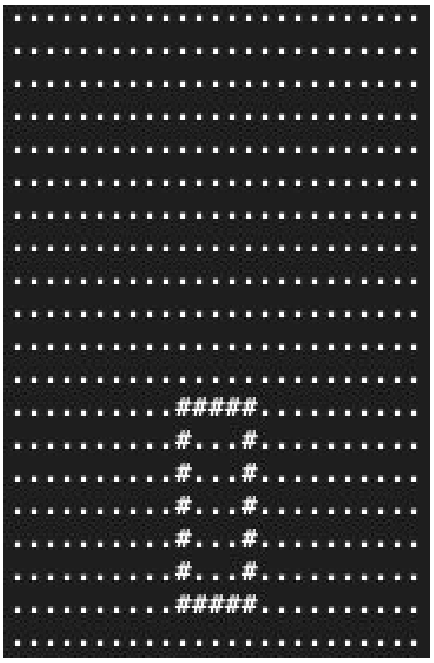
ptlDrawRect will draw a rectangle, the parameters to pass are a ptlRaster, a char that the rect will be made of, an int for the width of the rectangle, an int for its height, and finally an int for the x and an int for the y position. note that the x and y position you pass mean the position of the left top corner of the rectangle.

code:



```c
#define PTLIB_IMPL
#include "ptlib.h"

int main(){
    ptlRaster screen;
    screen = ptlInitRaster(25, 20, '.');

    ptlDrawRect(screen, '#', 5, 7, 10, 12);

    ptlRepaint(screen);

    ptlDestroyRaster(screen);

    return 0;
}
```

result:

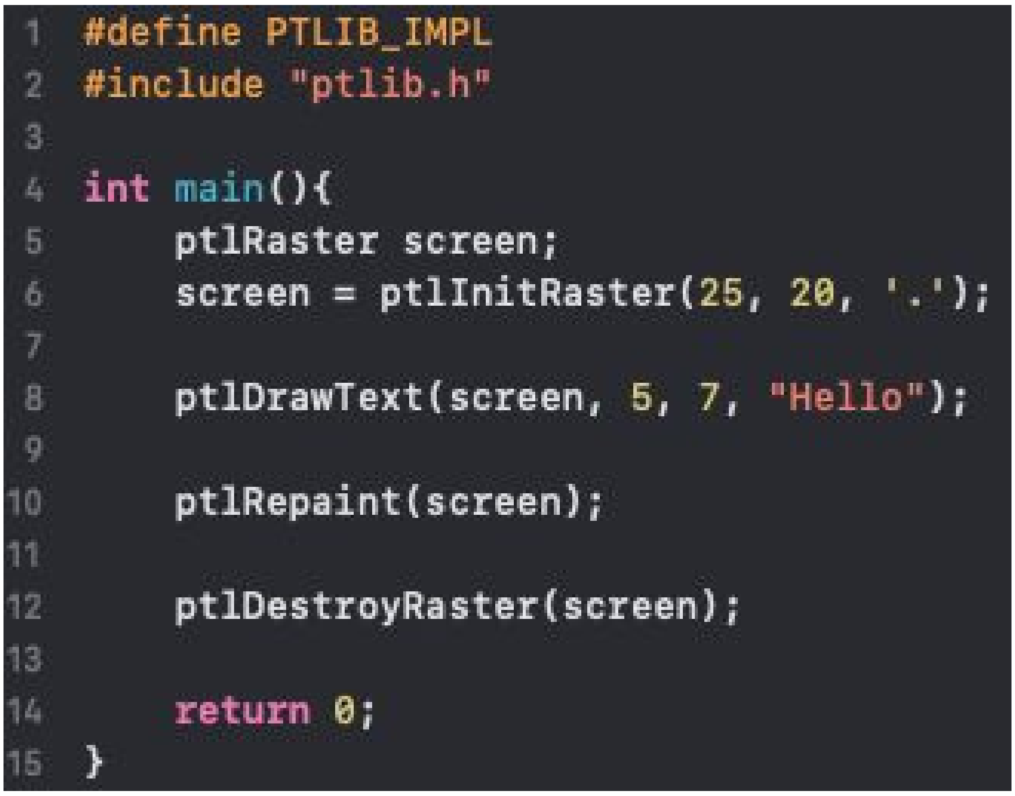


```
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
..........#####..........
..........#...#..........
..........#...#..........
..........#...#..........
..........#...#..........
..........#...#..........
..........#####..........
.........................
```

5 - ptlDrawText():

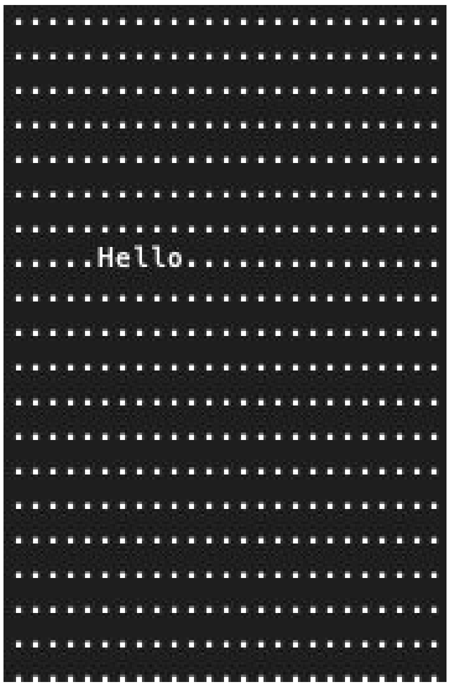
ptlDrawText will draw a string to the raster. the function takes the following parameters:
a ptlRaster, an int for the x and an int for the y position position at which the text should start, and of course a string that it has to draw.

code:



```c
#define PTLIB_IMPL
#include "ptlib.h"

int main(){
	ptlRaster screen;
	screen = ptlInitRaster(25, 20, '.');

	ptlDrawText(screen, 5, 7, "Hello");

	ptlRepaint(screen);

	ptlDestroyRaster(screen);

	return 0;
}
```

result:



```
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.....Hello...............
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
```