PTLIB DEVELOPER MANUAL

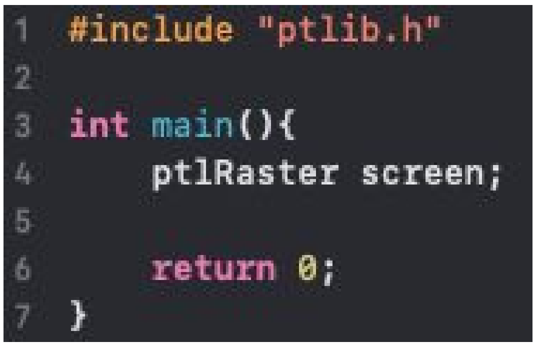
When including the ptlib.h file, you’ll get access to simple functions for the command line that abstract away boilerplate code in C. Those functions serve two purposes:

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.

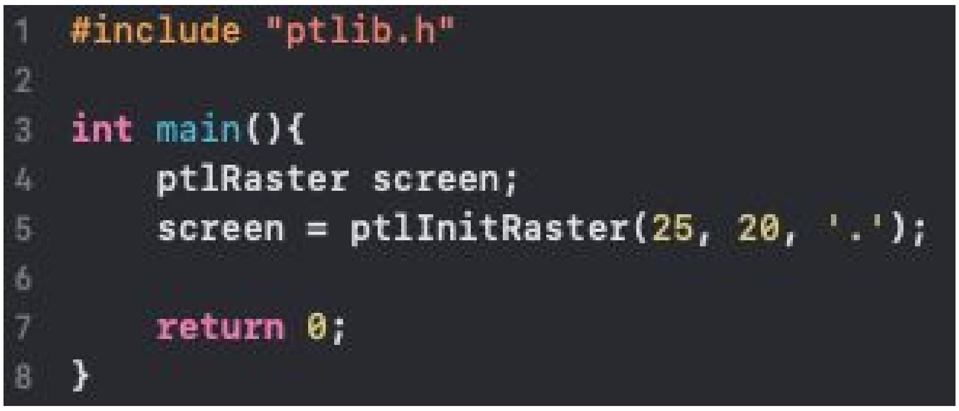
the raster:

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.



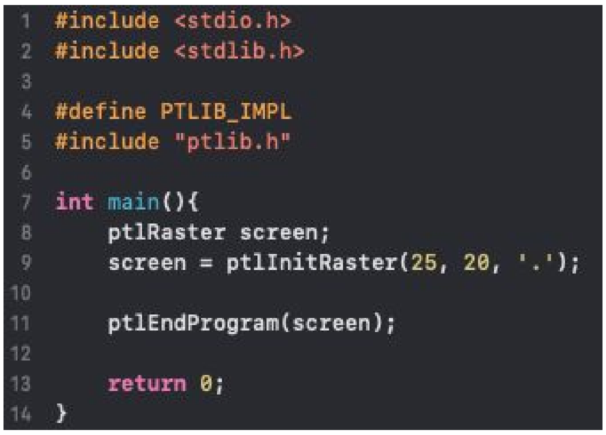
```c
#include "ptlib.h"

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.



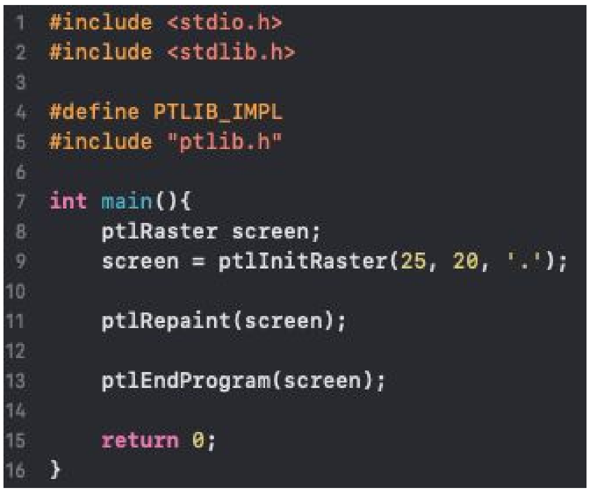
```c
#include "ptlib.h"

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 ptlEndProgram() in your code, if you don't, then the terminal window input settings will be messed up after the program has finished.



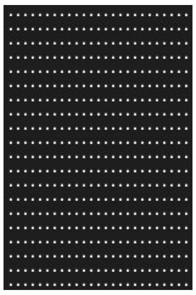
```c
#include <stdio.h>
#include <stdlib.h>

#define PTLIB_IMPL
#include "ptlib.h"

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

    ptlEndProgram(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 ptlEndProgram() 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
#include <stdio.h>
#include <stdlib.h>

#define PTLIB_IMPL
#include "ptlib.h"

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

    ptlRepaint(screen);

    ptlEndProgram(screen);

    return 0;
}
```

if we run that we'll see a raster with a width of 25 characters (or pixels) and a height of 20 characters. those characters are the '.' character. This is the result:

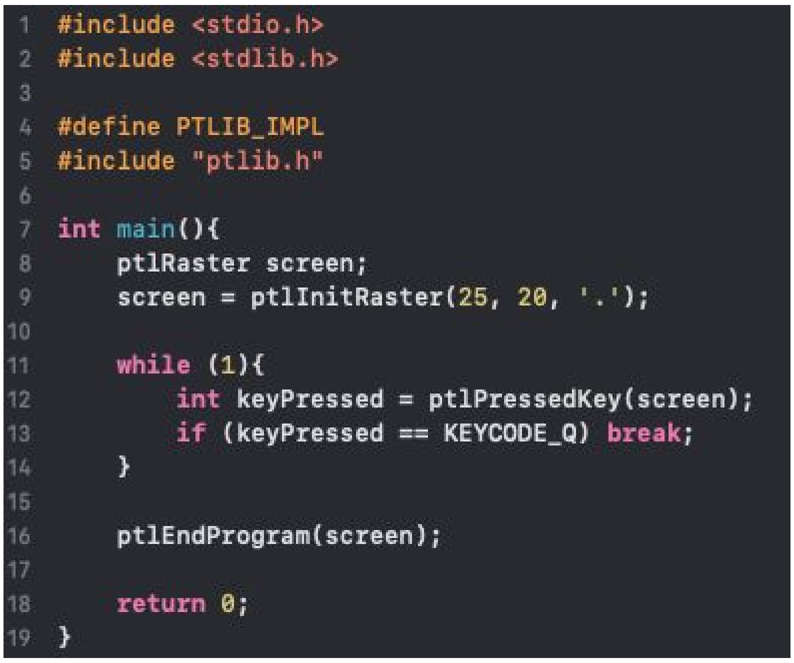


```
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................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

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 doesn’t only hold information about the graphics, but also about the inputs. We can compare the return value of ptlPressedKey() 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;
}
```

it is highly recommended to only call ptlPressedKey() only **ONCE** each iteration and store that return value in an integer. You should pretty much always do this for various reasons which we won’t go into in this part of the manual. However, if you really want what happens behind the scenes and why you should stick with this method of getting inputs, you can scroll all the way to the bottom where you’ll find explanations for that in the “behind the scenes” section.

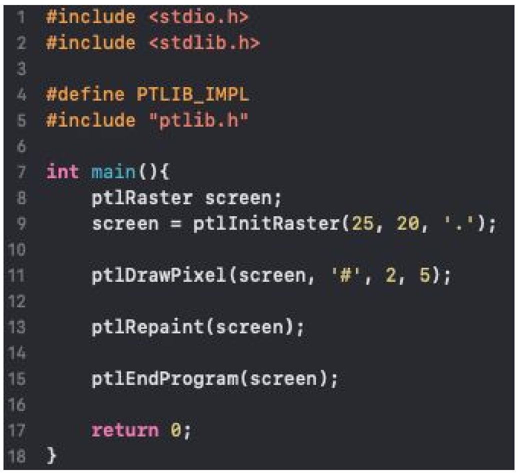
graphics:

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. Here's a quick overview of the graphics drawing functions:

1 - ptlDrawPixel():

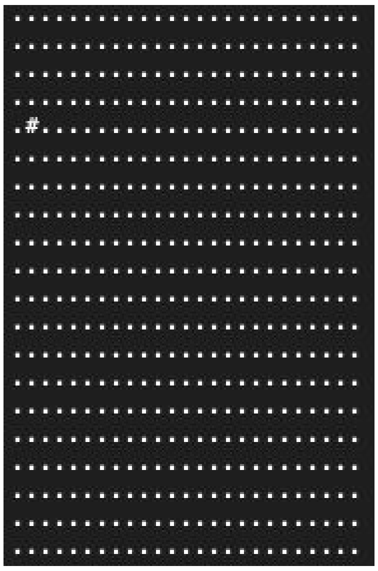
ptlDrawPixel does what it looks like: it draws a pixel in a raster, it takes a raster to draw the pixel in, a value on the x axis, a value on the y axis and a char to draw at that coordinate.

code:



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

#define PTLIB_IMPL
#include "ptlib.h"

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

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

    ptlRepaint(screen);

    ptlEndProgram(screen);

    return 0;
}
```

result:

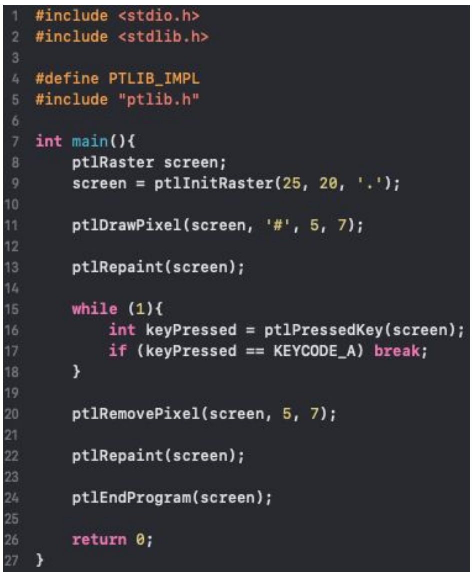


```
.........................
.........................
.........................
.........................
.#.......................
.........................
.........................
.........................
.........................
.........................
.........................
.........................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

2 - ptlRemovePixel():

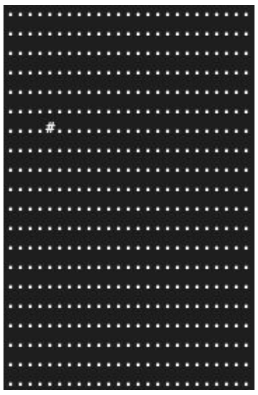
ptlRemovePixel will remove a pixel (or character) at a certain coordinate, and change the character at that coordinate to the background character. You could also use ptlDrawPixel() and pass it the background character of the raster, but that'd look uglier than just using ptlRemovePixel.

code:



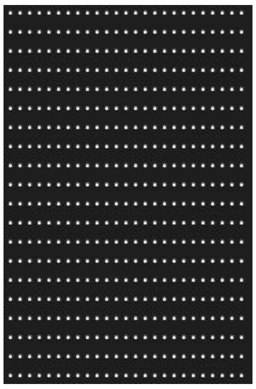
```c
#include <stdio.h>
#include <stdlib.h>

#define PTLIB_IMPL
#include "ptlib.h"

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

    ptlDrawPixel(screen, '#', 5, 7);

    ptlRepaint(screen);

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

    ptlRemovePixel(screen, 5, 7);

    ptlRepaint(screen);

    ptlEndProgram(screen);

    return 0;
}
```

result (before pressing "a"):



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

result (after pressing "a"):

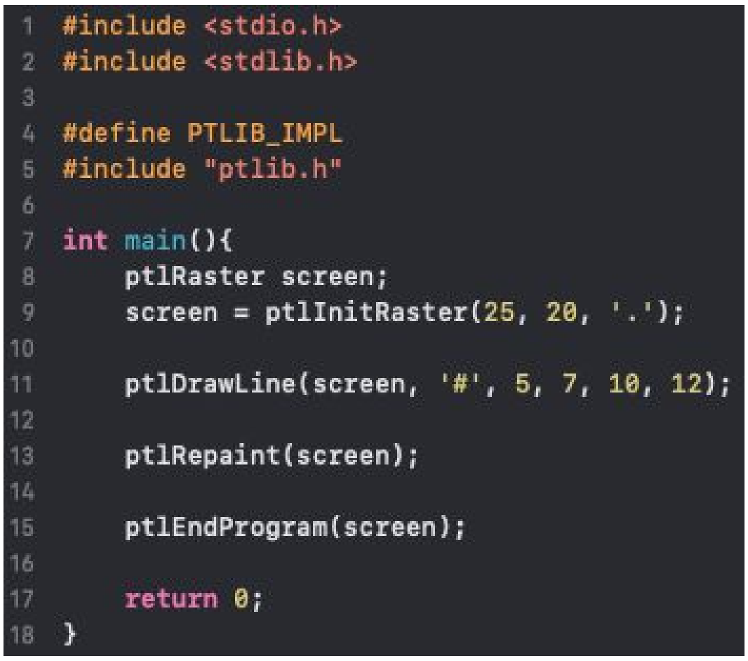


```
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................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

3 - ptlDrawLine():

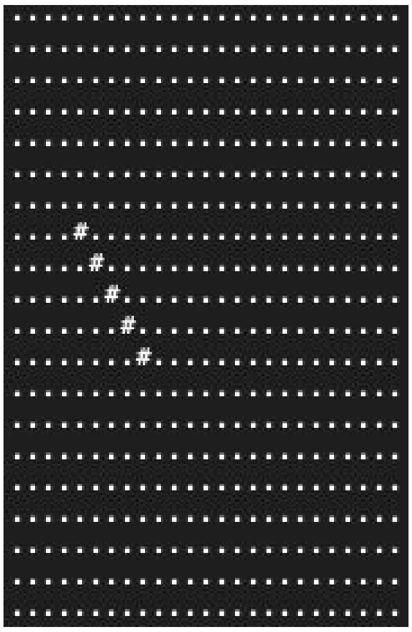
ptlDrawLine will draw a line from one point to another, it takes quite a lot of parameters: ptlRaster, pixelChar(the char of which the line will be made), start_x, start_y, end_x and end_y. Here’s what it looks like:

code:



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

#define PTLIB_IMPL
#include "ptlib.h"

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

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

    ptlRepaint(screen);

    ptlEndProgram(screen);

    return 0;
}
```

result:

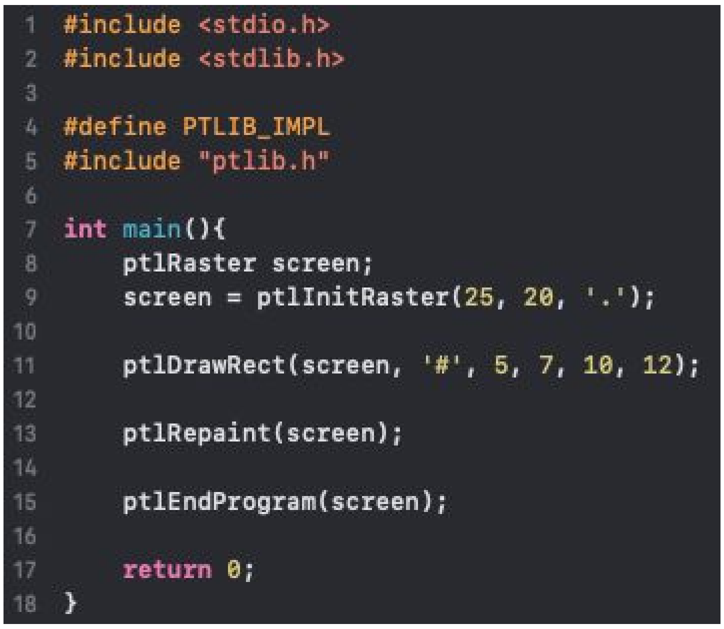


```
.........................
.........................
.........................
.........................
.........................
.........................
.........................
....#....................
.....#...................
......#..................
.......#.................
........#................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

4 - ptlDrawRect():

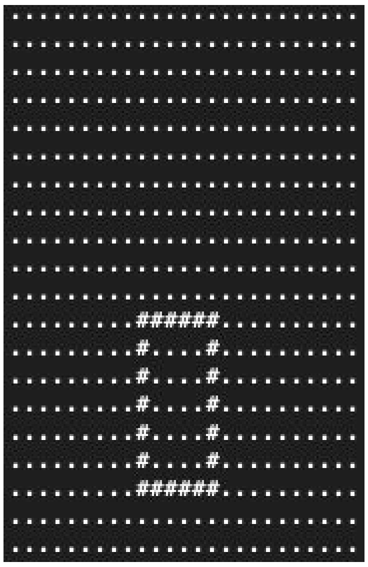
ptlDrawRect will draw a rectangle, the parameters to pass are a ptlRaster, a char that the rect will be made of, an int for the width of the rectangle, an int for its height, and finally an int for the x and an int for the y position. note that the x and y position you pass mean the position of the left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

code:



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

#define PTLIB_IMPL
#include "ptlib.h"

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

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

    ptlRepaint(screen);

    ptlEndProgram(screen);

    return 0;
}
```

result:

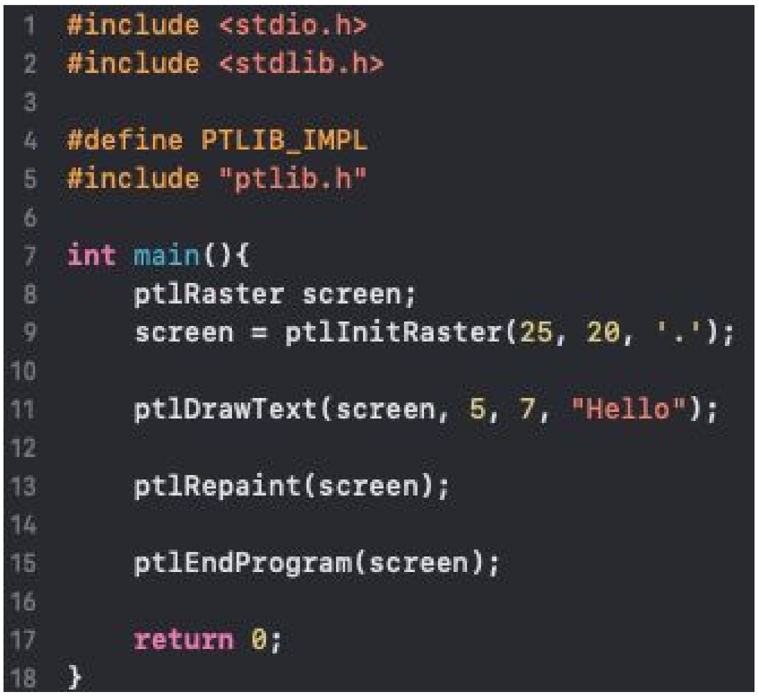


```
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.........######..........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

5 - ptlDrawText():

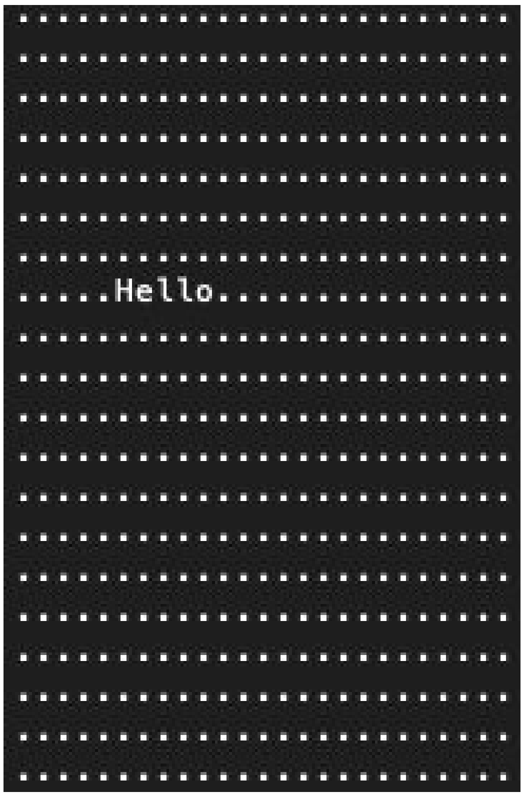
ptlDrawText will draw a string to the raster. the function takes the following parameters:
a ptlRaster, an int for the x and an int for the y position position at which the text should start, and of course a string that it has to draw.

code:



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

#define PTLIB_IMPL
#include "ptlib.h"

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

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

    ptlRepaint(screen);

    ptlEndProgram(screen);

    return 0;
}
```

result:



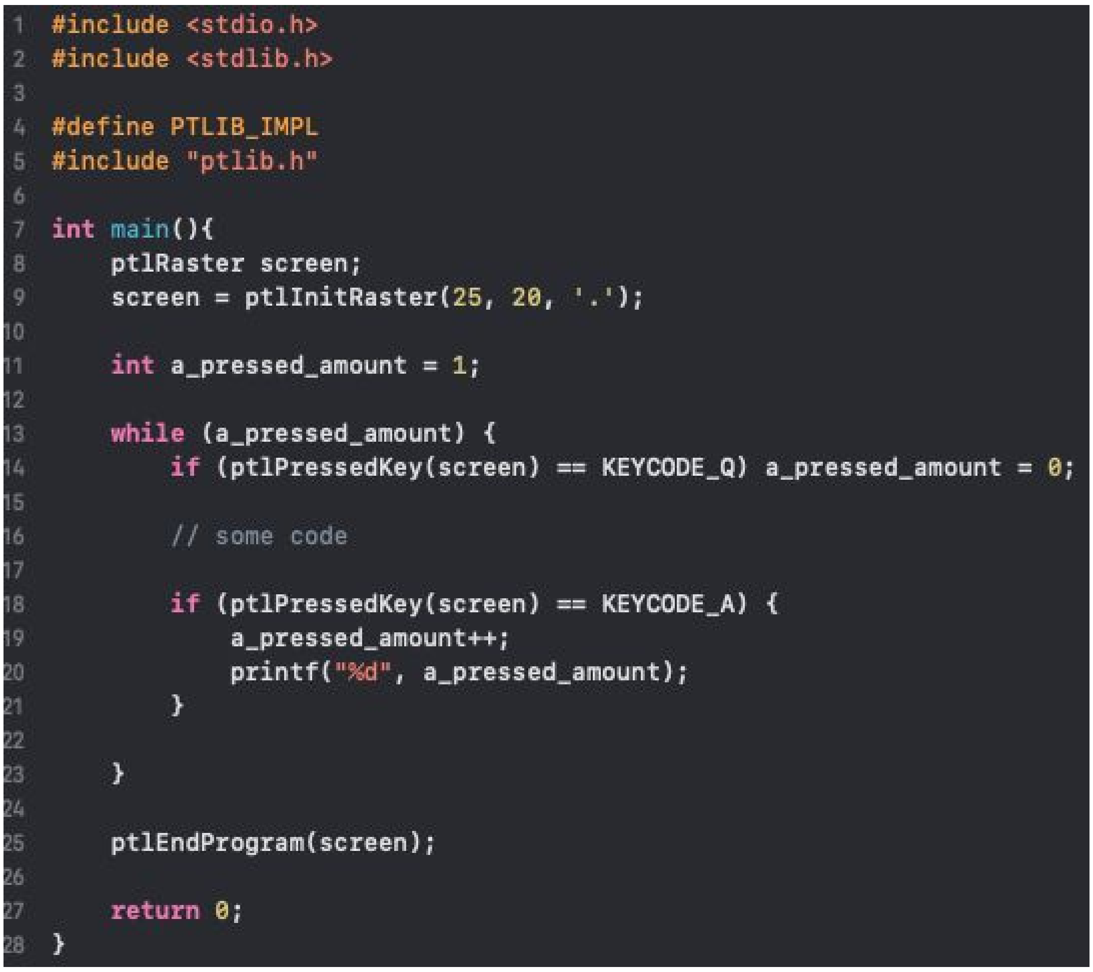
```
.........................
.........................
.........................
.........................
.........................
.........................
.........................
.....Hello...............
.........................
.........................
.........................
.........................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

BEHIND THE SCENES:

If you got this far, then you're either really curious about why you should handle input in the way it was said, or you just mindlessly scrolled down to the bottom of the page. Either way, you're a legend!

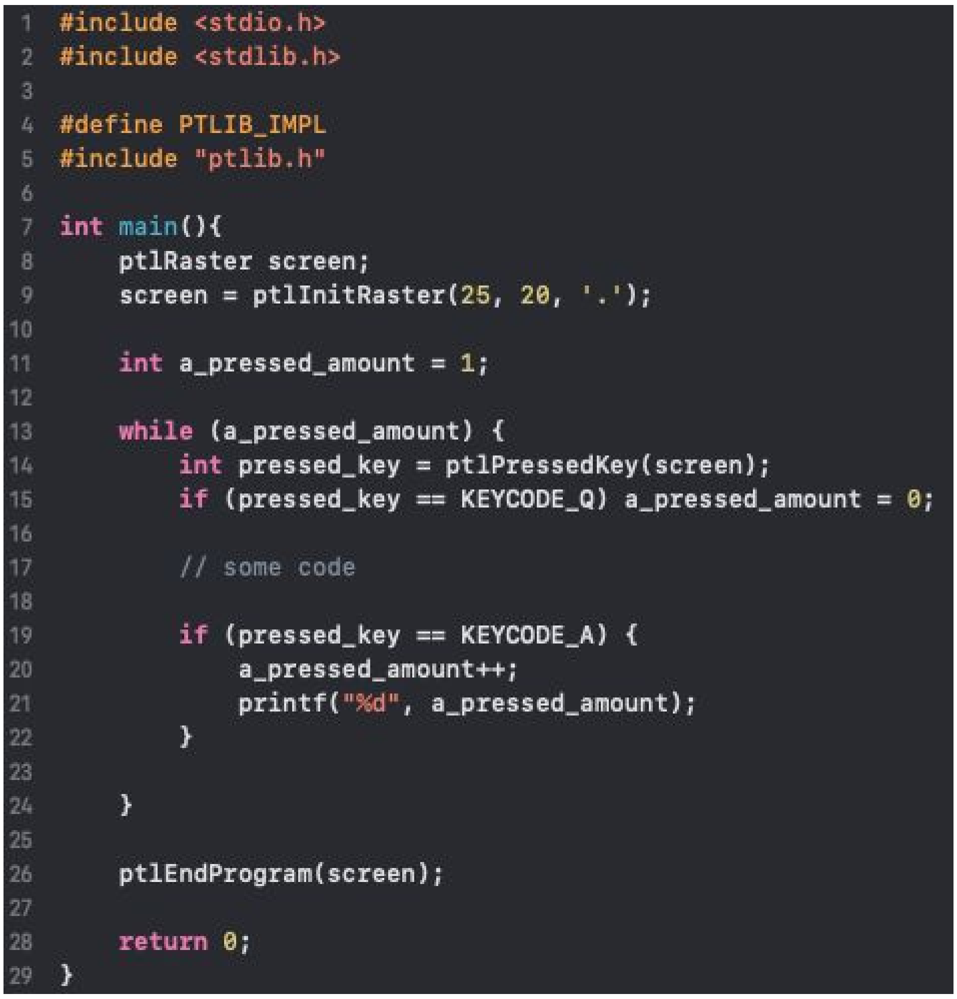
so about that input thing, the way input is handled behind the scenes is that on a different thread there's a function waiting to read some input from the terminal i/o, and as soon as it reads a character, the character is sent to the ptlRaster struct you initialized. It does this over and over again. the thread starts when you call ptlInitRaster(), and it "joins" or "stops" when you call ptlEndProgram. Because these characters are being read on a different thread, it won't care if there are two different characters being sent by the user and read by the program in one iteration of your while loop. So if you've got bad luck, it's perfectly possible for the user to press different keys during the same iteration, and that's often a very bad thing.

here is an example:



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

#define PTLIB_IMPL
#include "ptlib.h"

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

    int a_pressed_amount = 1;

    while (a_pressed_amount) {
        if (ptlPressedKey(screen) == KEYCODE_Q) a_pressed_amount = 0;

        // some code

        if (ptlPressedKey(screen) == KEYCODE_A) {
            a_pressed_amount++;
            printf("%d", a_pressed_amount);
        }

    }

    ptlEndProgram(screen);

    return 0;
}
```

This is obviously an absolutely useless program, but it's purpose is to show what can happen. as you can see the program might quit after the user has pressed "q", but if the user was the press "a" right after, it's possible that the program won't end. this is normally not what you want. so that is why the recommended way of using ptlPressedKey() is like this:



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

#define PTLIB_IMPL
#include "ptlib.h"

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

    int a_pressed_amount = 1;

    while (a_pressed_amount) {
        int pressed_key = ptlPressedKey(screen);
        if (pressed_key == KEYCODE_Q) a_pressed_amount = 0;

        // some code

        if (pressed_key == KEYCODE_A) {
            a_pressed_amount++;
            printf("%d", a_pressed_amount);
        }

    }

    ptlEndProgram(screen);

    return 0;
}
```

in this case ptlPressedKey() is only called once each iteration, and so the outcome won't depend on if the user pressed "a" quickly enough after pressing "q".

That was all there is to know about ptlib, now you can start to easily make cool cross-platform command line programs!