

Programando para PalmOs desde sistemas libres (en C)

Javier M Mora (jamarier)

15 de septiembre de 2006

Índice

1. Introducción	3
1.1. Recuperando la información	4
1.2. Características de una PDA con PalmOs	5
1.3. Versatilidad	6
1.3.1. Hacks	7
1.3.2. Notificaciones	8
1.3.3. PilotMain	9
1.4. Memoria	11
1.5. Asignación de Memoria	15
1.6. Por qué programar	15
2. El proceso y las herramientas	16
2.1. Registro en PalmSource	17
2.2. Proceso a vista de pájaro	17
2.3. pilot-template	18
2.4. pilrc	19
2.4.1. Los recursos	21
2.5. Descarga de las librerías de PalmOs	22
2.6. ROM para pose	23
2.7. Depurando	24
3. Programando una aplicación: Bola8	26
3.1. Definiendo el programa	27
3.2. Empleando la plantilla	27
3.3. Finalizando	30

4. Index

31

2a `<beamerProgramacionPalmC.tex 2a>`≡
`<cabecera fichero latex 2b>`
`<introducción 3a>`
`<herramientas y proceso 16g>`
`<bola8 26b>`
`<anexos 31>`

2b `<cabecera fichero latex 2b>`≡ (2a)
`\documentclass[] {beamer}`
`%\documentclass[ignorenonframetext] {beamer}`
`%\documentclass {article}`
`%\usepackage {beamerarticle}`
`\usepackage [latin1] {inputenc}`
`\usepackage [T1] {fontenc}`
`%\usepackage {thumbpdf}`
`\usepackage [spanish] {babel}`
`\usepackage {ae,aecompl}`
`\usepackage {listings}`
`%\usepackage {graphicx}`

`\title {Programando para PalmOs desde sistemas libres \\ (en C)}`
`\author {Javier M Mora (jamarier)}`
`\date {\today}`

`\begin {document}`
`%\newcommand {\fdiapositiva} [1] { \frame {falta: #1 } }`
`\newcommand {\fdiapositiva} [1] { falta: #1 }`

`\lstset {numbers=left,language=C}`

2c `<y ahora 2c>`≡ (3a 16g 26b)
`\begin {frame}`
`\frametitle {Y ahora...}`
`\tableofcontents [currentsection,hideothersubsections]`
`\end {frame}`

1. Introducción

```
3a <introducción 3a>≡ (2a)
    <portada 3b>
    <recuperar informacion 4a>
    <tabla de contenidos 4b>
    <y ahora 2c>
    \section{Introducción a las PDA con Palm Os}
    \subsection{Características}
    <características 5a>
    <versatilidad 6>
    \subsection{Memoria}
    <memoria 11a>
    <asignación dinámica 15a>
    \subsection{Piensa antes de empezar a programar}
    <por qué programar 15b>
    <fin primera parte 16f>

3b <portada 3b>≡ (3a)
    \section*{Inicio}
    \frame{\titlepage}
```

Hola, soy Javier M Mora (jamarier), soy casi ingeniero tec. industrial de Sevilla y voy a hablar de mis pensamientos y peleas sobre esto de la programación para PalmOs.

Cada vez que se organiza una jornada de Badopi, hay miedo por parte de la organización de que se presente mucha gente. Así que una de las técnicas que emplean para evitar aglomeraciones es programar actividades horripilantes y así desanimar a la gente a que venga.

Si, lo habeis acertado, esta es una de esas actividades. Antes de empezar con esta charla, me gustaría saber la proporción Usuarios de PDA, de PalmOs y programadores de C que hay en la sala. Para ello, los poseedores de un aparato PalmOs (zires, tungstens, treos, clies) levanten la mano derecha. Aquellos que sepan algo de C levanten la mano izquierda, si saben mucho que la levanten mucho y si además programan algo para PalmOs que den una palmada. Si alguien se dedica profesionalmente a esto, que se levante y hable el por mí. El que tenga un PocketPC, de un zapatazo.

1.1. Recuperando la información

```
4a <recuperar informacion 4a>≡ (3a)
\frame{
\frametitle{¿Dónde recuperar información de esta charla?}

\begin{itemize}
\item \url{http://barbacana.net/javierm/palmbadopi2006}
\item \url{http://del.icio.us/jamarier/palm+badopi}
\end{itemize}
}
```

Todo el contenido de la charla está disponible para bajarselo en <http://barbacana.net/badopi> Igualmente los enlaces con comentarios están en <http://del.icio.us/jamarier/palm+badopi>.

Y ahora pasemos a ver de que va esta charla viendo como está estructurada:

```
4b <tabla de contenidos 4b>≡ (3a)
\frame{
\frametitle{Contenidos}
\tableofcontents[hidesubsections]
}
```

La charlita está preparada en tres partes. Una primera donde vamos a ver las características de la plataforma, sus compromisos a la hora de diseñar que nos darán sus fuerza y su limitación.

En una segunda parte, veremos el proceso y las herramientas que se emplean a la hora de programar

Y en la última parte, si seguimos vivos habrá un ejemplo de programación en directo.

1.2. Características de una PDA con PalmOs

Antes de ver la programación en sí debemos de ver cuales son las características de diseño de estas PDA. Dichas características nos van a dar el fuerte y las debilidades de la plataforma y es importante para un desarrollador conocerlas.

Las características de la palm que van a delimitar la forma de trabajo con ella son: <http://www.palmos.com/dev/support/docs/palmos/PalmOSCompanion/Nutshell.html#995827>

```
5a <características 5a>≡ (3a) 5b>
    \begin{frame}
    \frametitle{Características de diseño de las PDAs con PalmOs\footnote{
    \url{http://www.palmos.com/dev/support/docs/palmos/PalmOSCompanion/Nutshell.html\#995827}}
    }}
```

Portabilidad , es pequeño ligero, puede llevarse en cualquier parte consigue la casi ubicuidad. Pantalla pequeña, es imperativo que la transferencia de información muy usuario/PDA esté muy optimizada. Limita otros recursos hardware: tamaño memoria, presencia de HD, otros periféricos.

Interactividad Ha de estar siempre disponible para funcionar. No puede tardar ni en arrancar de apagado ni en ejecutar ningún programa.

```
5b <características 5a>+≡ (3a) <5a 5c>
    \includegraphics<2>[width=\linewidth]{caracteristicas1.png}%
```

Autonomía , duración de las baterías elevadas. Muchísimo comparado con la duración actual de un portatil. El precio que hay que pagar, no son muy potentes.

```
5c <características 5a>+≡ (3a) <5b 5d>
    \includegraphics<3>[width=\linewidth]{caracteristicas2.png}%
```

Versatilidad , Apta para muy variada cantidad de actividades. Flexibilidad para hacer cosas para la que no fue diseñada. hacks de roms. Wiki y compañía. O el espejo o ...

```
5d <características 5a>+≡ (3a) <5c 5e>
    \includegraphics<4>[width=\linewidth]{caracteristicas3.png}%
```

Compatibilidad , todos los dispositivos son compatibles entre sí (en características comunes).

```
5e <características 5a>+≡ (3a) <5d
    \includegraphics<5>[width=\linewidth]{caracteristicas4.png}%
    \includegraphics<6>[width=\linewidth]{caracteristicas5.png}%
    \end{frame}
```

incluso tenemos cierta garantía de que si en un futuro aparece un nuevo dispositivo PalmOs, será compatible. (porque ya lo ha hecho en el pasado; una muestra de esto es Micros y demás. Sistema PACE de compatibilidad arm y m68k.

Robusto Ha de ser un sistema que no se bloquee disponible bajo cualquier situación (y que no pierda datos ni se bloquee). El reset de la Palm solo borra la memoria dinámica. Dado que los datos están en la zona estática no hay pérdida de datos en un reset.

Estas características limitan las capacidades del dispositivo. Una potencia reducida y los interfaces de usuarios han de estar muy depurados. (Tomado de la url de antes y sin traducir).

Conectividad PC connectivity is an integral component of the Palm Powered handheld

Entrada de datos Therefore, you should not require users to enter a lot of data on the handheld itself.

Potencia If your application needs to perform a computationally intensive task, you should implement that task in the desktop application instead of the handheld application.

1.3. Versatilidad

Hablemos ahora de la versatilidad del dispositivo.

```
6 <versatilidad 6>≡ (3a)
\begin{frame}
\frametitle{Versatilidad}
\begin{itemize}[<+>]
\item Sistema monotarea. Cuando tu programa se ejecuta, tiene control absoluto sobre la máquina.
\item \hyperlink{hacks}{Hacks de roms \beamergotobutton{}}
\item \hyperlink{eventloop}{Orientado a eventos (público) \beamergotobutton{}}
\item \hyperlink{pilotmain}{Delegación de responsabilidad hacia el programa/programador \beamergotobutton{}}

\end{itemize}

\hypertarget<2>{cualidades2}{}
\hypertarget<3>{cualidades3}{}
\hypertarget<4>{cualidades4}{}

\end{frame}
```

1.3.1. Hacks

```
7a <hacks 7a>≡ (31) 7b>  
\begin{frame} %%%hay que rehacer las imágenes  
\frametitle{Hacks}  
\includegraphics<1>[width=\linewidth]{problem1.png}
```

Estas imágenes no son mias, sino que proceden de 79bmedia y les pedí permiso para incluirlas y me contestaron:

Sure, go ahead. A simple link (and maybe mentioning Crash Pro and Skinner ;-)) will be more than sufficient.

La idea es la siguiente: Los servicios que implementa PalmOs no están enlazados directamente desde el programa que los usa, sino que emplea una tabla de direccionamiento. Cuando llamamos a la función que borra la pantalla, el sistema mira la dirección real en la tabla y la ejecuta.

Dicha tabla está en memoria Ram, así que podemos alterarla a gusto ;-)

```
7b <hacks 7a>+≡ (31) <7a  
\includegraphics<2>[width=\linewidth]{problem2.png}  
  
\url{http://www.79bmedia.com/content/view/83/73/lang,en/}  
\hypertarget{hacks}{}  
\hyperlink<2>{cualidades2}{\beamerreturnbutton{}}  
\end{frame}
```

En este caso, se ha alterado la función por una propia que puede borrar las líneas impares primero y luego las pares. En el caso del diagrama, la función «pirata» llama después a la original. Porque no queremos perder funcionalidad, pero no siempre ha de ser así.

Este sistema de actualización era el empleado en Palm antiguas. Ahora se ha establecido un modo más sencillo aún de modificación del sistema.

1.3.2. Notificaciones

```

8  <notificaciones 8>≡ (31)
    \begin{frame}[containsverbatim]
    \frametitle{Bucle de Eventos}
    \begin{lstlisting}
    static void EventLoop(void)
    {
        Word err;
        EventType e;

        do {
            EvtGetEvent(&e, evtWaitForever);
            if (! SysHandleEvent (&e))
                if (! MenuHandleEvent (NULL, &e, &err))
                    if (! ApplicationHandleEvent (&e))
                        FrmDispatchEvent (&e);
        } while (e.eType != appStopEvent);
    }
    \end{lstlisting}
    \hypertarget{eventloop}{}
    \hyperlink{cualidades3}{\beamerreturnbutton{}}
    \end{frame}

```

Esta es la función del bucle principal del programa, el gestor de eventos. (No hay que preocuparse en escribir esta función porque existen sistemas automáticos para hacerlo) El funcionamiento es el siguiente: el comando `EvtGetEvent` espera hasta que se produzca un evento. En un programa «normal», se espera indefinidamente hasta que dicho evento se produzca. En programas que tengan animaciones o que se quiera que la PDA haga algo de fondo se usará un límite de tiempo en la espera de eventos.

Hay distintos tipos de eventos y todos pasan por aquí. Así, si se pulsa con el lápiz en la pantalla se produce un evento de pantalla pulsada. Al procesarse ese evento, el sistema puede darse cuenta que se ha pulsado sobre un botón, en cuyo caso se añade un evento de botón pulsado en la cola de eventos y se da por procesado el evento actual.

El evento va probandose en cascada. Primero la gestión de eventos del sistema y posteriormente los del programa. Aunque nada nos impide alterar el orden de los eventos si hiciese falta. Por ejemplo, para atrapar una pulsación del botón de apagado.

En el `while` vemos que se comprueba para un tipo de evento determinado y termina el bucle.

1.3.3. PilotMain

Aquí tenemos la función `PilotMain` por defecto. No hay que saber programarla porque también se genera automáticamente.

A la función `PilotMain` se la llama con tres parámetros. El primero es el comando de ejecución, el segundo un puntero a los parámetros adicionales y el tercero son banderas que definen el tipo de ejecución.

Lo interesante es que un programa puede ser llamado/ejecutado por muy diversos motivos y dicho motivo va en `cmd`. Por ejemplo, cuando se busca algo, no es `PalmOs` el que se encarga de buscar la información. `PalmOs` se encarga de arrancar todos los programas instalados pasándole el comando de búsqueda y el texto a buscar. Y es cada programa el que ha de buscar.

`PalmOs` por tanto deriva la responsabilidad de buscar a cada programa.

Este sistema de arranque permite una comunicación sencilla entre aplicaciones de la `palm`. Pudiendo llamar a otras aplicaciones con comandos sencillos.

```

9  <pilotmain 9>≡ (31) 10>
    \begin{frame}[containsverbatim]
    \frametitle{PilotMain básico}
    \begin{lstlisting}
    /* Main entry point */
    DWord PilotMain(Word cmd, Ptr cmdPBP,
                    Word launchFlags)
    {
        Word err;

        if (cmd == sysAppLaunchCmdNormalLaunch) {
            err = StartApplication();

```

```

        if (err) return err;

        EventLoop();
        StopApplication();

    } else { return sysErrParamErr; }

    return 0;
}
\end{lstlisting}

\hypertarget{pilotmain}{}
\end{frame}

```

Existen un gran número de comandos de ejecución disponibles, además de los que el programador puede definir para su uso personal.

Es importante respetar los comandos empleados pero hay libertad para usos «creativos» de los mismos. Es posible, por tanto, desobedecer a PalmOs. Por ejemplo, en vez de buscar, se puede hacer una estadística de aquellos datos que son más buscados. (Cuando recibe el parametro de buscar, en vez de buscar, apuntar el término buscado en una base de datos).

10 \langle *pilotmain 9* \rangle +≡ (31) <9

```

\begin{frame}
\frametitle{Extracto de Comandos de ejecución de programas PalmOs}
\begin{itemize}
\item sysAppLaunchCmdAddRecord
\item sysAppLaunchCmdAlarmTriggered
\item sysAppLaunchCmdFind
\item sysAppLaunchCmdGoTo
\item sysAppLaunchCmdNormalLaunch
\item sysAppLaunchCmdSyncNotify
\item sysAppLaunchCmdSystemReset
\end{itemize}

\url{http://www.palmos.com/dev/support/docs/palmos/PalmOSCompanion/AppStartupAndStop.html}

\hyperlink{cualidades4}{\beamerreturnbutton{}}
\end{frame}

```

sysAppLaunchCmdAddRecord Solicita que se añada un registro nuevo. Esto permite por ejemplo añadir citas a la agenda desde cualquier otro programa

sysAppLaunchCmdAlarmTriggered Se llama cuando se ha disparado una alarma

sysAppLaunchCmdFind Se le pasa una cadena de texto para buscar

sysAppLaunchCmdGoTo Se pide que se abra el programa y que muestre inmediatamente el registro indicado

sysAppLaunchCmdNormalLaunch Arranque normal.

sysAppLaunchCmdSyncNotify Se llama cuando ha sido un éxito la sincronización.

sysAppLaunchCmdSystemReset Se llama tras un evento de reset.

1.4. Memoria

```
11a <memoria 11a>≡ (3a) 11b>
    \begin{frame}
    \frametitle{Distribución de la memoria\footnote{
    \url{http://www.palmos.com/dev/support/docs/palmos/PalmOSCompanion/Memory.html}}
    }}
    \includegraphics<2>[width=\linewidth]{memoria1.png}%
```

El único sistema de almacenamiento de la PDA es la memoria. Sin tener en cuenta los dispositivos que tienen posibilidad de usar tarjetas de memoria externa flash, toda la memoria que tenemos es simple y llanamente memoria RAM volátil. Cuando la PDA está supuestamente apagada, las baterías siguen alimentando a la memoria para que no pierda los datos ni los programas (por eso la batería de los PDA se gastan aunque estén apagados). (Para escribir en la memoria flash es necesario picos importantes de tensión/intensidad que le quitarían autonomía).

Dicha memoria se destina a dos funcionalidades distintas: memoria dinámica o heap y la memoria estática.

```
11b <memoria 11a>+≡ (3a) <11a 11c>
    \includegraphics<3>[width=\linewidth]{memoria2.png}%
```

La memoria dinámica o heap que representa aproximadamente un 1/32 del total de la memoria. (la representación está a a escala)

```
11c <memoria 11a>+≡ (3a) <11b 12a>
    \includegraphics<4>[width=\linewidth]{memoria3.png}%
```

O dicho de otra forma en mi pda son 256k¹.

En la memoria estática o de almacenamiento se almacenan el código ejecutable y los datos de las aplicaciones.

12a `<memoria 11a>+≡ (3a) <11c 12b>
\includegraphics<5>[width=\linewidth]{memoria4.png}%`

En caso de un reset blando de la memoria, los contenidos de esta zona de memoria no se alterarán.

12b `<memoria 11a>+≡ (3a) <12a 12c>
\includegraphics<6>[width=\linewidth]{memoria5.png}%`

En la heap se almacena las variables de ejecución del programa, la pila, las tablas de hacks, asignación dinámica, las variables globales...

Voy a hacer una pequeñísima digresión. Al programar en C, la memoria se organiza en zonas de memoria llamadas chunks. ******escribir la palabra chunk en la diapositiva 6******

12c `<memoria 11a>+≡ (3a) <12b 12d>
\includegraphics<7>[width=\linewidth]{memoria6.png}%`

Un chunk es una zona de memoria continua que tienen el mismo origen y la misma finalidad. Para PalmOs, el tamaño máximo de un chunk son 64k menos una cabecera de información del mismo (donde por ejemplo se indica el tamaño del mismo).

Continuamos con el uso de la memoria:

El modo de almacenamiento de información en la zona estática es la base de datos. Una base de datos es un chunk de memoria que apunta a otros chunks de memoria que son los registros y que pueden tener otros chunks de datos de configuración y parámetros externos a los registros. Son de formato libre.

Gráficamente:

12d `<memoria 11a>+≡ (3a) <12c 13a>
\includegraphics<8>[width=\linewidth]{memoria7.png}%`

¹Es muy poca memoria dinámica por lo que el primer objetivo para realizar cualquier programa en PDA es conservar la memoria del heap

Tenemos una base de datos que es un chunk. Dicho chunk tiene una serie de datos en una estructura. Y tiene una tabla que apunta a otros chunks que contienen los datos.

Aunque se llame base de datos, no es una base de datos estructurada en el sentido que estamos acostumbrados. Cada base de datos solo contiene una tabla.

Las bases de datos tienen que tener nombre único en la PDA, así que es práctica común darle un nombre que incluya el nombre del programa o el CreadorID como prefijo. En caso contrario podrían existir dos aplicaciones que intentasen escribir en una base de datos llamada `users` corromper los datos uno de otro.

Cada base de datos tiene dos identificadores: Creador ID y Typo ID. Al borrar un programa se borran automáticamente todas las bases de datos con el Creador ID que tiene el programa. De esta forma se asegura una desinstalación limpia de los programas. No pueden coexistir dos programas en una misma Palm con idéntico Creador ID, de intentar instalar algo así, se supone una actualización y se machaca el programa antiguo. programa².

De ahí la importancia de que no existan dos aplicaciones con igual Creador ID y la obligación de registrar nuestras aplicaciones con su Creador ID en la web de palmdev: <http://www.palmos.com/dev/creatorid/> (esto lo veremos en la parte práctica). Hay identificadores reservados para el uso de Palm.

El TipoId, es equivalente a la extensión de los ficheros. Sirve para tener nuestras bases de datos clasificadas. Dado que cada BD tiene solo una tabla, para relaciones complejas puede ser necesario tener varias BD enlazadas. Podemos señalar cada tabla con un tipo distinto. Hay algunos tipos que están reservados por Palm para usos especiales.

13a `<memoria 11a>+≡ (3a) <12d 13b>
\includegraphics<9>[width=\linewidth]{memoria8.png}%`

Puede, la base de datos tener un chunk de información común. Por ejemplo, en una base de datos clasificación de la liga, el nombre de los equipos iría aquí.

13b `<memoria 11a>+≡ (3a) <13a 14a>
\includegraphics<10>[width=\linewidth]{memoria9.png}%`

²<http://www.palmos.com/dev/support/docs/palmos/PalmOSCompanion/Nutshell.html#996041>

Y por supuesto, luego tendríamos cada registro guardado en un chunk distinto. Como dije antes, estas bases de datos no son lo que estamos acostumbrados a ver con el SQL. Cada registro sólo contiene un campo (un array de bytes). Y es misión del programador crear una función que convierta sus datos a un array y otra para recuperar la información guardada sobre dicho array. Normalmente, la información de todos los registros es homogénea. Suele estar formado por una estructura (un struct) de tamaño predeterminado con información sobre los campos y tamaños de los datos reales que van a continuación. Osea cabecera + datos auténticos.

Es decir, en el ejemplo anterior de la liga, los resultados de los partidos. Y no en un registro resultados de partidos, en otro, nombres de jugadores, en otro estadios, etc.

Un programa también es una base de datos cuyo contenido es código su TipoID es obligatoriamente ".app". (crear la base de datos es tarea de las pre-tools y nosotros no nos vamos a ocupar de ello).

14a `<memoria 11a>+≡ (3a) <13b 14b>
\includegraphics<11>[width=\linewidth]{memoriaA.png}%`

Llegamos al último punto interesante reseñar referente al uso de la memoria. Las asignaciones dinámicas.

En esta imagen tenemos un puntero que apunta al inicio de un chunk. La memoria dinámica es escasa y el chunk (por el motivo que sea) está situada en medio de una zona vacía de memoria. Así que estamos limitando el tamaño máximo de la memoria que podemos usar.

PalmOS tiene un dispositivo de gestión de memoria que permite mover un chunk con su contenido, a aquella parte que le resulte más conveniente. Lamentablemente, todavía no hace milagros. Y no puede controlar los punteros que apuntan a aquella zona. (si mueve el chunk el puntero pasará a apuntar a ninguna parte y un puntero loco no es deseable).

Así que para solucionar esto se recurre a un método similar a los hacks ya comentado: los punteros indirectos. Llamados en este caso «Handles»

14b `<memoria 11a>+≡ (3a) <14a 14c>
\includegraphics<12>[width=\linewidth]{memoriaB.png}%`

Un Handle es un puntero que apunta a un puntero que apunta al chunk. Este último puntero está situado en una zona de memoria controlada por PalmOS. De forma que cuando mueve un chunk, cambia este último puntero.

Claro que hay que buscar un sistema para evitar que se mueva un chunk cuando estamos trabajando con él. Y no podemos trabajar con el último puntero porque estaríamos en el mismo caso de antes.

El mecanismo propuesto es bloquear el handle.

14c `<memoria 11a>+≡ (3a) <14b
\includegraphics<13>[width=\linewidth]{memoriaC.png}%
\end{frame}`

*******hay que cambiar el color del handle cuando se bloquea*******

Cuando queremos asegurarnos de que un chunk no se mueva bloqueamos el handle que lo apunta "Lock", usamos el puntero correspondiente, y desbloqueamos el handle.

Un handle se puede bloquear hasta 14 veces simultáneamente. Usa un Lock-count de 4 bits.

Hay un tercer sistema (además de los punteros y los handles) para referenciar chunks en la memoria estática pero no lo vamos a tratar hoy. (LocalID y CardID).

1.5. Asignación de Memoria

```
15a <asignación dinámica 15a>≡ (3a)
\begin{frame}
\frametitle{Asignación de memoria
dinámica\footnote{\url{http://www.palmos.com/dev/support/docs/palmos/PalmOSCompanion/Memoria}}
\framesubtitle{Principales funciones}
\begin{description}[<+>]
\item[MemPtr MemPtrNew(size\_t numbytes)] equivalente a malloc. Reserva
un chunk de memoria no movil.
\item[MemHandle MemHandleNew(size\_t numbytes)] Reserva un chunk de
memoria movil.
\item[MemPtr MemHandleLock(MemHandle it)] Impide el movimiento de un
chunk e incrementa el contador de bloqueo. Devuelve un puntero a dicha zona.
\item[MemHandleUnlock(MemHandle) o MemPtrUnlock(MemPtr)] Decrementa el
contador de bloqueo y desbloquea si llega a cero.
\item[MemPtrFree(MemPtr) y MemHandleFree(MemHandle)] libera la
memoria (a cada uno el que corresponda). Bloquear un Handle no asigna
nueva memoria por lo que dicho puntero no hay que liberarlo.
\end{description}
\end{frame}
```

1.6. Por qué programar

Ya conocemos un poco mejor nuestra plataforma y antes de empezar a programar habría que plantearse. ¿Cuándo es necesario programar? O mejor dicho, cuando no merece la pena programar una Palm.

```
15b <por qué programar 15b>≡ (3a) 15c>
\begin{frame}
\frametitle{¿Por qué programar para una PDA?}
\includegraphics<2>[width=\linewidth]{noprogramar1.png}%
```

1. Aparece una necesidad (propia o de un cliente).

```
15c <por qué programar 15b>+≡ (3a) <15b 16a>
\includegraphics<3>[width=\linewidth]{noprogramar2.png}%
```

2. No se cubre con soluciones gratuitas, abiertas ni libres

```
16a <por qué programar 15b>+≡ (3a) <15c 16b>
      \includegraphics<4>[width=\linewidth]{noprogramar3.png}%

      http://www.freewarepalm.com/
```

```
16b <por qué programar 15b>+≡ (3a) <16a 16c>
      \includegraphics<5>[width=\linewidth]{noprogramar4.png}%
```

3. No se cubre con soluciones de pago y/o son muy caras

```
16c <por qué programar 15b>+≡ (3a) <16b 16d>
      \includegraphics<6>[width=\linewidth]{noprogramar5.png}%

      http://www.palmgear.com/
```

```
16d <por qué programar 15b>+≡ (3a) <16c 16e>
      \includegraphics<7>[width=\linewidth]{noprogramar6.png}%
```

4. Se hace uno el programa (o lo encarga).

```
16e <por qué programar 15b>+≡ (3a) <16d
      \includegraphics<8>[width=\linewidth]{noprogramar7.png}%
      \end{frame}
```

```
16f <fin primera parte 16f>≡ (3a)
      \begin{frame}
      \frametitle{Felicidades}
```

```
Has conseguido aguantar hasta el fin de la primera parte (de tres).
\end{frame}
```

2. El proceso y las herramientas

En esta segunda parte, vamos a describir el ciclo de trabajo típico de una aplicación. Si queda alguien vivo al terminarla haremos en la tercera parte un ejemplo en vivo de una aplicación siguiendo los pasos descritos.

```
16g <herramientas y proceso 16g>≡ (2a)
      \section{El proceso y las herramientas}
      <y ahora 2c>
      \subsection{Registro en Palm}
      <registro en palmsource 17a>
      \subsection{Proceso a vista de pájaro}
      <proceso a vista de pájaro 17e>
      \subsection{Depuración}
      <depuración 24a>
      <fin de la segunda parte 26a>
```

Como primer paso, antes de hablar del programa en sí, consiste en registrarse en la web de palm source.

2.1. Registro en PalmSource

Es un paso imprescindible porque es la única forma de que nuestras aplicaciones no tienen un CreadID que interactúe con otra aplicación. Además de que son las únicas fuentes legales de las librerías. (la rom, que sería el tercer elemento que descargaremos de palmsource se puede descargar de nuestra propia palm, si es que tenemos).

Lamentablemente, no puedo dar una url de acceso. (son sesiones privadas con urls no aprovechables).

```
17a <registro en palmsource 17a>≡ (16g) 17b>
    \begin{frame}[t]
      \frametitle{Registrándose en PalmSource}
```

```
      \url{http://www.palmsource.com/developers/}
```

En esta página hay que ir a la sección indicada como Members Area.

```
17b <registro en palmsource 17a>+≡ (16g) <17a
    \begin{center}
      \includegraphics[scale=0.3]{palmsource1.png}%
    \end{center}
    -> Members Area
  \end{frame}
```

Si uno no está registrado puede registrarse en este momento y al finalizar volver a este punto.

De vez en cuando al entrar, te hacen una encuesta. Y pasemos ahora al proceso

2.2. Proceso a vista de pájaro

```
17c <cabecera proceso 17c>≡ (17e 19a 21 22c)
    \begin{frame}
      \frametitle{Proceso}
```

```
17d <end proceso 17d>≡ (18a 19a 21 23a)
    \end{frame}
```

Vamos a ver los elementos necesarios en el proceso de creación de un programa.

```
17e <proceso a vista de pájaro 17e>≡ (16g) 18a>
    <cabecera proceso 17c>
    \includegraphics<2>[width=\linewidth]{proceso1.png}%
```

Este es el esquema general. Partimos de unos ficheros fuente, un fichero de recursos y ¡gracias a dios! de un Makefile que nos permite juntar todo sin muchos quebraderos de cabeza. Y obtener nuestro ejecutable.

Realmente el fichero .prc es, como ya dije, una base de datos que incluye el ejecutable y los recursos. (en dos minutos hablamos de los recursos)

```
18a <proceso a vista de pájaro 17e>+≡ (16g) <17e 18b>
    \includegraphics<3>[width=\linewidth]{proceso2.png}%
    <end proceso 17d>
```

2.3. pilot-template

```
18b <proceso a vista de pájaro 17e>+≡ (16g) <18a 19a>
    <pilot-template 18c>
```

```
18c <pilot-template 18c>≡ (18b) 18d>
    \begin{frame}
    \frametitle{pilot-template\footnote{pilot-template -h}}
    pilot-template [-pbitm] prcname appname iconname CrID
```

Este comando nos crea un prc de nombre prcname, cuya aplicación se llama appname y cuyo nombre que aparece en la navegación es iconname. También hay que proporcionar el CreadorID (estos parámetros se pueden luego editar en el Makefile generado).

El programa nos genera los siguiente ficheros.

```
18d <pilot-template 18c>+≡ (18b) <18c>
    \begin{description}
    \item<2>[uno.c] con main, bucle de eventos, y carga de primer
    formulario.
    \item<2>[uno.rcp] formulario inicial (vacío), enlace al icono del programa e indicador de
    \item<2>[unoRcs.h] puente entre los números de recursos y nombres más
    manejables.
    \item<2>[callback.h] definición de macro para evitar un bug de gcc
    \item<2>[uno.pbm] icono inicial.
    \item<2>[Makefile] fichero para automatizar la construcción del prc
    final.
    \end{description}
    \end{frame}
```

Si se quiere añadir más ficheros fuente, solo hay que tocar el Makefile para que considere nuestros nuevos objetivos.

Al ver las capacidades de los PDA vimos el bucle del gestor de eventos y el PilotMain. Entonces dije que eran códigos que se repiten casi intactos en todos los proyectos. La utilidad pilot-template se encarga de generar un esqueleto de aplicación con estas funciones pre-programadas. Y nosotros nos encargamos a partir de aquí de llenar de contenidos donde interese.

```
19a <proceso a vista de pájaro 17e>+≡ (16g) <18b 19b>
    <cabecera proceso 17c>
    \includegraphics<1>[width=\linewidth]{proceso2.png}%
    \includegraphics<2>[width=\linewidth]{proceso3.png}%
    <end proceso 17d>
```

Ya es hora de hablar de los recursos: El fichero de recursos nos permite separar toda la parte visual del programa (los botones, los mensajes, los textos, las ventanas...) de su procesado.

Las ventajas de separar son:

- Tener centralizado todo el aspecto gráfico en vez de repartido por el código. Los recursos están en un formato más compacto que el C (más información importante concentrada).
- Permite tener dos roles trabajando en el código: un diseñador y programador
- Facilita la labor de traducción, ya que todas las cadenas de texto están centralizadas.

Los recursos están en un formato de texto y es necesario procesarlo a una forma binaria más eficiente. Esta operación la efectúa la utilidad pilrc. Cuando hablemos de pilrc veremos ejemplo de formato de los recursos.

2.4. pilrc

```
19b <proceso a vista de pájaro 17e>+≡ (16g) <19a 21b>
    <pilrc 20>
```

20

`<pilrc 20>≡`

(19b)

```
\begin{frame}
\frametitle{pilrc}
\begin{itemize}
\item<1->Documentación en file:///usr/share/doc/pilrc/html/manual.html (ojo, Debian)
\item<2->Existen herramientas para simplificar la escritura:
  \begin{description}
    \item[pilrcui] visualizador (aproximado) de rcp sin compilar
    \item[guikachu] programa visual de generación de rcp (cuyo paquete no funciona en Debian)
  \end{description}
\item<3->La gestión de pilrc es transparente gracias a Makefile
\item<4->\hyperlink{rcp}{Los recursos son identificados como números.
En fichero unoRsc.h con \#defines los convertimos en constantes más
manejables. \beamergotobutton{}}
\end{itemize}

\hypertarget<4>{pilrc3}{}
\end{frame}
```

pilrcui tiene problemas con el tamaño de las fuentes y no es visualización perfecta. guikachu depende de cierta librerías de python antiguas y no se instala actualmente en debian.

2.4.1. Los recursos

21a \langle recursos 21a $\rangle \equiv$ (31)

```

\begin{frame}[containsverbatim]
\frametitle{Nombrando recursos}
\begin{block}{uno.rcp}
\begin{lstlisting}
#include "unoRsc.h"
FORM UnoForm 0 0 160 160
BEGIN
    TITLE "Uno"
    BUTTON "Nuevo" ID NuevoCanBut AT (10 10 45 12)
    NOGRAFFITISTATEINDICATOR
END
\end{lstlisting}
\end{block}
\begin{block}{unoRsc.h}
\begin{lstlisting}
#define UnoForm          100
#define NuevoCanBut     10001
\end{lstlisting}
\end{block}

\hyperlink{pilrc3}{\beamerreturnbutton{}}
\hypertarget{rcp}{}

\end{frame}

```

21b \langle proceso a vista de pájaro 17e $\rangle + \equiv$ (16g) \langle 19b 21c \rangle

```

\begin{block}{cabecera proceso 17c}
\includegraphics<1>[width=\linewidth]{proceso3.png}%
\end{block}
\end{proceso 17d}

```

Ya estamos terminando los elementos o pasos necesarios para tener nuestro programa. Para optimizar espacio, tal y como se dijo, la plataforma no tiene librería estándar de C. En su defecto está la librería de Palm. (prc-tools si tienen librería estándar para palm pero el coste de tamaño que proporciona usar un «printf» es muy elevado).

21c \langle proceso a vista de pájaro 17e $\rangle + \equiv$ (16g) \langle 21b 22a \rangle

```

\begin{block}{cabecera proceso 17c}
\includegraphics<1>[width=\linewidth]{proceso4.png}%
\includegraphics<2>[width=\linewidth]{proceso5.png}%
\end{block}
\end{proceso 17d}

```

2.5. Descarga de las librerías de PalmOs

22a `<proceso a vista de pájaro 17e>+≡ (16g) <21c 22c>`
`<descarga de librerías 22b>`

Estas librerías están disponibles en la web para descargarse en la página de palmdev. Lamentablemente, en las condiciones de uso de palm no se permite la redistribución. Así que es estrictamente necesario bajarlo personalmente de su página.

Dado que no existe control sobre el contenido de dicho fichero, y ya ha ocurrido que han modificado la organización de directorios dejando inusables las `prc-tools`, Una vez bajado es necesario escanear dichas librerías para ajustar las tools a las librerías. La herramienta `palmdev-prep` se encarga de ello (esta herramienta solo hay que ejecutarla cada vez que se bajen nuevas versiones).

Voy a ser un poco rápido porque no merece la pena pararse mucho aquí.

22b `<descarga de librerías 22b>≡ (22a)`

```
\begin{frame}
  \frametitle{Descarga de librerías}
  \begin{enumerate}
    \item -> \url{http://www.palmsource.com/developers/}
    \item -> Members Area (e introducción de contraseña)
    \item -> Palm Os Developer Tools
    \item -> Core Palm Os SDK
    \item -> Palm OS Garnet SDK (68K) R3 PRC tools Generic UNIX
    \item (se descarga palms-sdk-5.0r3-1.tar.gz.tar)
    \item Descompresión en /usr/local/share/palmdev
    \item (crea un directorio dentro de palmdev llamado sdk-5r3)
    \item \# palmdev-prep
  \end{enumerate}
\end{frame}
```

Garnet es el nombre en clave para la última remesa de PDAs disponibles. Desde las Zire en adelante. A partir de la versión 5.3 de la Rom de Palm Os. Cobalt es el nuevo sistema disponible en prototipos. Versión 6 de Palm Os

Puede pedir en algún momento que «firmes» el acuerdo de licencia en mi caso como ya lo firme, ahora no me lo ha pedido y no recuerdo exactamente cuando te lo preguntan.

Por defecto, `palmdev-prep` busca la librería en la localización dada. Así que nos eliminamos muchos quebraderos de cabeza si está donde se busca.

22c `<proceso a vista de pájaro 17e>+≡ (16g) <22a 23a>`
`<cabecera proceso 17c>`
`\includegraphics<1>[width=\linewidth]{proceso5.png}%`
`\includegraphics<2>[width=\linewidth]{proceso6.png}%`

Con esto y ejecutando make, nos generará el deseado fichero prc.

Una vez generado el fichero prc. Ya se puede pasar a la fase de pruebas que aquí he representado por la simulación, depurado con un debugger o transferencia real a un dispositivo físico.

```
23a <proceso a vista de pájaro 17e>+≡ (16g) <22c 23b>
      \includegraphics<3>[width=\linewidth]{proceso7.png}%
      <end proceso 17d>
```

Estas operaciones se efectúan con las utilidades POSE (pose - Palm OS Emulator), que es un emulador. gdb que es el debugger emparejado con de gcc. Y finalmente lo transferimos con la utilidad que corresponda. (tambien está kpilot, jpilot, gpilot.....)

2.6. ROM para pose

Para usar POSE es necesario tener una rom de un dispositivo Palm. Esta se puede descargar de un dispositivo palm que tengamos o bajarse una de la zona de desarrolladores de Palm.

```
23b <proceso a vista de pájaro 17e>+≡ (16g) <23a>
      <rom pose 23c>
```

```
23c <rom pose 23c>≡ (23b) 23d>
      \begin{frame}
      \frametitle{ROM para POSE}
      \begin{enumerate}
      \item<1-> -> \url{http://www.palmsource.com/developers/}
      \item<1-> -> Members Area (e introducción de contraseña)
      \item<1-> -> Platform ROMs
      \item<1-> (nos bajamos la versión debug de windows: PalmOS412\_DbgROMs.zip)
```

Existen dos tipos de Roms. La versión «Release» que es una copia de las ROMs de los dispositivos físicos y la versión «Debug» que incluye información de depuración que nos simplificará la vida a la hora de depurar nuestro programa.

Además, palm distingue entre una versión Inglesa exclusiva y otra llamada EFIGS (English, French, Italian, German, Spanish), que permite cambiar el idioma de los programas. Nosotros usaremos la versión multilingüe.

Para tener las cosas más o menos ordenadas dejamos las roms en /usr/local/share/palmdev

```
23d <rom pose 23c>+≡ (23b) <23c>
      \item<2-> descomprimos en /usr/local/share/palmdev
      \item<3-> Hay que crear al menos un perfil en Pose.
      \end{enumerate}
      \end{frame}
```

Un perfil es un fichero que relaciona una rom, con un tipo de máquina emulada, con una cantidad de memoria. (y así)

2.7. Depurando

Sería un estúpido si dijera que la fase de depuración no es necesaria.

```
24a <depuración 24a>≡ (16g) 24b>
\begin{frame}
\frametitle{Depuración 1/2}
\framesubtitle{POSE}
\begin{itemize}
\item<1-> Ejecuta nuestra aplicación y podemos interaccionar con ella
\item<2-> Controla errores "de segundo nivel" de nuestra aplicación
\begin{itemize}
\item Accesos a memoria no reservada
\item Memoria no liberadas
\item Intentos de escritura de ROM
\item Pila exhausta
\item Accesos a Handles no bloqueados
\item ...
\end{itemize}
\end{itemize}
```

La clasificación de error de segundo nivel es mía. Los errores de primer nivel son los errores de sintaxis que pilla el compilador. Los de segundo nivel se refieren a errores lógicos que aunque compilen bien tienen un fallo de programación por olvido o despiste. por ejemplo, olvidarse de liberar la memoria asignada dinámicamente al final de una función. El programa funciona, pero si se llama suficientes veces a la misma, obtendremos un error de memoria agotada. Los de tercer nivel son fallos de diseño algoritmos o planteamientos que pensamos buenos y que son malos.

```
24b <depuración 24a>+≡ (16g) <24a 24c>
\item<3-> Gremlins
```

Los Gremlins también son una herramienta a nuestro favor. Es una utilidad incluida en pose que genera pseudoaleatoriamente millones de pulsaciones en nuestra aplicación. Activando menús y pulsando botones. Escribiendo textos de Shakespeare y similares. Ofrece tal estrés a los programas que podemos suponer que si supera los gremlins no tiene errores. Y son pseudoaleatorios porque nos permite reproducir las secuencias de interacción para poder localizar donde se produce nuestro error.

```
24c <depuración 24a>+≡ (16g) <24b 25a>
\item<4-> pose-profile (versión que hace estadísticas de las funciones
del programa
```

Hay una versión de pose que permite hacer un profile (analizar estadísticamente los tiempos usados por cada función) y así establecer políticas de optimización.

```
25a <depuración 24a>+≡ (16g) <24c 25b>
    \item<5-> pose -load\_apps uno.prc (es tu amigo)
    \end{itemize}
\end{frame}
```

Dado que pose emula una palm cada vez que queremos probar una nueva versión de nuestro programa tenemos que hacer la emulación de la transferencia. Cuando uno depura mucho es un poco pesado. Hasta que descubrí que se puede hacer la transferencia automáticamente al arrancar pose.

No voy a hablar de gdb en general, porque si habeis trabajado con gcc, lo habreis tocado en alguna ocasión. Recordar que funciona en consola, que existen front-end gráficos como ddd para simplificar su uso, que puede hacer depurado simbólico y podemos ver nuestras líneas de código fuente en vez de ensamblador.

```
25b <depuración 24a>+≡ (16g) <25a 25c>
    \begin{frame}
    \frametitle{Depuración 2/2}
    \framesubtitle{gdb}
    \begin{itemize}
    \item<1-> m68k-palms-gdb (debian)
    \item<2-> Es cruzado, necesita un sistema anfitrión (pose)
```

Cuando empecé a usar turbo-Debugger, pensé que los Debugger eran una especie de simuladores que eran capaces de simular la ejecución de cada instrucción de nuestro programa. Aunque sigo sin saber mucho de tecnología de debuggers, ahora sé que se puede configurar al micro para que lance una interrupción tras cada línea de código ejecutado. Esa interrupción es controlada por el programa debugger que permite examinar el flujo del programa original.

Cuando se programa en C para un PC, el debugger puede ejecutar en un hilo el programa a examinar y por otro lado examinar el estado del programa. Este sistema no puede efectuarse en caso de compilación cruzada ya que el PC no puede ejecutar directamente el programa para Palm. Afortunadamente, POSE permite establecerse como anfitrión y se intercomunica con gdb para poder hacer la depuración.

```
25c <depuración 24a>+≡ (16g) <25b>
    \item<3> .gdbinit al rescate:
    \begin{quote}
    file uno \\
    target pilot localhost:2000
    \end{quote}
    \end{itemize}
\end{frame}
```

.gdbinit se carga y ejecuta automáticamente cuando gdb arranca. El sitio ideal para ejecutar los comandos que siempre vamos a ejecutar. En este caso, cargar el fichero objeto que contiene información de depuración y enlazar con pose que es quien realmente realizará la simulación.

Por tanto, es importante ejecutar primero Pose y luego gdb. Yo tengo un script que lo carga en orden.

```
26a <fin de la segunda parte 26a>≡ (16g)
    \begin{frame}
    \frametitle{Albricias}
    Aquí termina la segunda parte. Ahora queda lo más divertido.
    \end{frame}
```

3. Programando una aplicación: Bola8

En esta última sección, vamos a programar una aplicación extremadamente compleja. Es un sistema experto de toma de decisiones: la bola8

```
26b <bola8 26b>≡ (2a)
    \section{Programando una aplicación}
    <y ahora 2c>
    \subsection{Definiendo el programa}
    <definición del problema 27a>
    \subsection{Generación de plantillas}
    <creando plantilla 27b>
    \subsection{Los Recursos}
    <escribiendo recursos 28>
    \subsection{El código}
    <escribiendo código 29>
    <finalizando 30>
```

3.1. Definiendo el programa

```
27a <definición del problema 27a>≡ (26b)
\begin{frame}
  \frametitle{Definiendo el programa}
  \begin{itemize}
    \item Vamos a contruir un sistema experto e inteligente de toma de
    decisiones: bola 8.

    \item Visualmente en la pantalla aparecera un botón y cada vez que se
    pulse aparecerá una respuesta a una pregunta formulada en voz alta
    cuya posible respuesta sea si o no. (Umm, reconocimiento de voz... ;-)

    \item Para simplificar el diseño, la longitud de las posibles respuestas
    será fijada en tiempo de compilación. Tendremos varias respuestas y
    mostraremos una al azar.
  \end{itemize}
\end{frame}
```

3.2. Empleando la plantilla

```
27b <creando plantilla 27b>≡ (26b)
\frame{
  \frametitle{Empleando la plantilla}
  \Large pilot-template bola8 bola8 bola8 BoLa
}
```

El CreaId es una cadena de 4 letras que ha de incluir al menos una mayúscula.
(este está ocasión no he registrado el identificador).

```
28 <escribiendo recursos 28>≡ (26b)
\begin{frame}[containsverbatim]
\frametitle{Escribiendo recursos}
\begin{lstlisting}
FORM MainForm 1 1 158 158
BEGIN
    BUTTON "Consulta" Id CONSULTA at ( CENTER 30 AUTO AUTO )
    FIELD Id RESPUESTA at ( 10 PREVBOTTOM+30 100 50 )
        NONEDITABLE MAXCHARS 22
END
\end{lstlisting}

\begin{lstlisting}
STRINGTABLE PosiblesRespuestas ""
"Seguro que si          "
"Seguro que no         "
"Quizás                "
"No lo aseguro         "
"Intentalo más tarde  "
";lo has pensado bien?"
"Sinceramente no lo sé"
\end{lstlisting}
\end{frame}
```

29 *<escribiendo código 29>*≡ (26b)

```

\begin{frame}[containsverbatim]
\frametitle{Escribiendo código}
\begin{lstlisting}
void *
getObjectPtr (FormPtr pForm, Int16 resourceNo)
{
    UInt16 objIndex=FrmGetObjectIndex(pForm,resourceNo);
    return FrmGetObjectPtr(pForm,objIndex);
}
\end{lstlisting}

\begin{lstlisting}
case CONSULTA:
{
    //no es estrictamente aleatorio ni justo
    int numRespuestaElegida= SysRandom(0) % numPosiblesRespuestas;
    Char cadenaRespuestaElegida[22]; //tamaño máximo de cadena + null
    ControlPtr FieldRespuesta=getObjectPtr(FrmGetActiveForm(),RESPUESTA);
    SysStringByIndex(PosiblesRespuestas,numRespuestaElegida,cadenaResp
staElegida,22);
    FldSetTextPtr(FieldRespuesta,cadenaRespuestaElegida);
    FldDrawField(FieldRespuesta);
}
\end{lstlisting}
\end{frame}

```

3.3. Finalizando

```
30 <finalizando 30>≡ (26b)
\section{Qué falta}
\begin{frame}
\frametitle{Qué falta}
\begin{itemize}
\item<2-> Mucho (Recursos)
\begin{itemize}
\item Elementos gráficos y edición "programática".
\item Animaciones
\item Internacionalización.
\end{itemize}
\item<3-> Mucho más (Gestión de periféricos)
\begin{itemize}
\item Sonido
\item Compact Flash o SD
\item Infrarrojos, Bluetooth, USB
\end{itemize}
\item<4-> Más aún (comunicaciones con PC)
\begin{itemize}
\item Conduits
\item Copias de seguridad de bases de datos
\item Operaciones cliente servidor
\end{itemize}
\end{itemize}
\end{frame}

\begin{frame}
\frametitle{Y ahora ¿qué?}
\begin{itemize}
\item<2-> Inmediatamente después de esta diapositiva toda mujer admiradora
(o no) que quiera puede acercarse y besar al ponente
\item<3-> He elaborado un liveCD con todas las utilidades necesarias.
Quien quiera puede acercarse y echarle un vistazo.
\item<4-> Si hay alguna pregunta en el tintero, puede hacerse ahora o
a lo largo de la jornada o incluso por correo (cobro poco).
\end{itemize}
\end{frame}

\begin{frame}
\frametitle{Fin}

{\Huge ;GRACIAS!}
```

```

(ahora chicas, este es el momento)
\end{frame}
****cambiar el trozo por el del código definitivo de bola8****
31 <anexos 31>≡ (2a)
    \appendix
    \section{Códigos fuente}

    <hacks 7a>
    <notificaciones 8>
    <pilotmain 9>
    <recursos 21a>

\end{document}

```

4. Index