IUP 入门指南

翻译:fangcun

2018 年 7 月 8 日

目录

1 概述

感谢阅读本教程。本教程的目的是为没有使用过 **IUP** 的同学提供一个快速入门的途径。首先，我们介绍什么是 **IUP** 以及它是如何帮助我们开发自己的程序。**IUP** 是一个**轻量级**, **跨平台**的用户界面接口。它提供了 **C/C++** 和 **Lua** 两种形式的 **API**。创建它的目的是为了用户界面代码可以不用修改就可以在不同的系统下编译。目前它支持的系统有:**GTK+**，**Motif** 和 **Windows**。使用 **IUP** 的好处还有: 高性能 [1] ，容易掌握[2]。**IUP** 使用**盒子**模型进行布局。

使用 **IUP**，我们首先需要理解它的三个主要概念: *元素*，*属性*, *回调*。

1.1 元素

*元素*就是程序中出现的各种界面元素。**IUP** 包含了许多用户界面元素。**IUP** 使用操作系统的原生界面元素。也就是说界面中按钮，文本框的绘制和管理是由操作系统进行的，界面元素的外观和同操作系统的其它程序是类似的。另一方面，这也造成了使用 **IUP** 的同一程序在不同操作系统下外观差异可能较大。除此之外，部分**控件**是由 **IUP** 自己绘制的。在 **IUP** 中，*对话框*用来代表各种类型的**窗口**。一个*对话框*可以包含一个或多个**控件**。

1.2 属性

*属性*用来改变和查询元素的参数。每个元素都包含一些*属性*来定义它们的行为和外观。不同元素的相同*属性*产生的影响可能是不同的。*属性*的名称总是使用大写。但*属性*的值是大小写无关的。[3]

1.3 回调

*回调*是当事件发生时，程序调用处理时间的**函数**。通常，*回调*在用户和界面元素交互时被调用。程序注册*回调*后，会在每一次事件发生时调用*回调*。

我们已经了解了 **IUP** 的一些概念。现在让我们开始从简单到复杂使用 **IUP** 构建图形界面程序。

2 Hello World

2.1 初始化

下面的代码显示了初始化 **IUP** 环境和使用消息对话框进行消息显示。每一行代码的解释在之后给出。

C语言

```
#include <stdlib.h>
#include <iup.h>

int main(int argc, char **argv)
{
  IupOpen(&argc, &argv);
```

[1] IUP 使用操作系统原生的用户界面，而不是自己进行绘制，所以用户界面的效率就是原生用户界面的效率。
[2] 它的 API 构成简单。
[3] 也就是说 *Yes*，*YES* 是一样的。

```
  IupMessage("Hello World 1", "Hello world from IUP.");

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")

iup.Message("Hello World 1","Hello world from IUP.")

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

首先，我们看 C 语言版本。在第一行我们包含了 C 语言的标准库，几乎所有 C 语言程序都需要使用它。接着我们包含我们的例子需要使用的 **IUP** 库，然后就是 *main* 函数。在执行 **IUP** 函数之前，我们需要使用 *IupOpen* 函数来初始化 **IUP**。下一行使用 *IupMessage* 函数创建一个消息对话框。*IupMessage* 函数接受两个参数:*title* 和 *message*。*title* 会被显示在对话框的标题，*message* 会被显示在对话框的窗体内。接着，我们调用 *IupClose* 函数来释放 **IUP** 使用的资源。最后，我们返回程序运行成功的标志。

接着是 **Lua** 版本。首先包含了我们需要使用的 Lua 模块。代码中的 *require* 等价于我们在 C 语言中使用的 *IupOpen* 函数。C 语言形式和 **Lua** 形式的最大不同表现在，在 C 中我们使用 **Iup** 前缀，在 **Lua** 中我们使用 **iup.**。也就是说 *iup.Message* 等价于 *IupMessage*，以此类推。另外，**Lua** 没有 *main* 函数的概念，它从第一行代码开始执行到最后一行代码。在 **Lua** 版本中，我们调用了 *iup.MainLoopLevel* 函数，它是为了检查消息循环是否已经在运行，来避免重复调用。这个函数只需要在子环境中使用，对于一般的程序不需要调用。

2.1.1 编译和连接

编译和连接使用 **IUP** 的程序需要指定头文件目录和库目录以及连接 **IUP** 库。下面给出一个命令行的例子：

```
gcc -o example2_1 example2_1.o
        -liupimglib -liup -L../../../lib/$TEC_UNAME
```

对于包含较多模块的程序，我们推荐使用 Makefile 进行处理。在 Linux 和 Windows 下也有一些非常不错的集成开发环境可以使用。它们的配置方法是类似的。

2.2 创建对话框

接下来我们修改第一个例子，添加我们自己的对话框。

C语言

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

int main(int argc, char **argv)
{
  Ihandle *dlg, *label;

  IupOpen(&argc, &argv);

  label = IupLabel("Hello world from IUP.");
  dlg = IupDialog(IupVbox(label, NULL));
  IupSetAttribute(dlg, "TITLE", "Hello World 2");

  IupShowXY(dlg, IUP_CENTER, IUP_CENTER);

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```lua
require("iuplua")

label = iup.label{title = "Hello world from IUP."}
dlg = iup.dialog{
  iup.vbox{label},
  title = "Hello World 2",
}

dlg:showxy(iup.CENTER,iup.CENTER)

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

可以发现，我们为 **IUP** 元素定义了 *Ihandle ** 变量。我们创建了两个不同的变量。一个是 *dlg*，表示我们的对话框，一个是 *label*。表示我们的文本信息。接下来我们通过 *IupDialog* 和 *IupLabel* 函数初始化了它们。*IupLabel* 函数的惟一参数就是标签的文本。*IupDialog* 函数的参数是一个 *Ihandle **，这里我们使用 *IupVbox* 函数作为它的参数。*IupVbox* 是用于垂直布局的盒子。*IupVbox* 函数接

受 *Ihandle ** 作为参数，并在自己的空间内垂直布局它们。在本例中，我们只有一个 *label* 空间，所以把它的第二个参数设置为 *NULL*。下一行，我们使用 *IupSetAttribute* 函数来修改控件属性。代码中，我们使用 *IupSetAttribute* 函数修改了对话框的标题。紧接着，我们调用 *IupShowXY* 函数来设置对话框显示在屏幕中央。接下来出现的是 *IupMainLoop* 函数。在其中 **IUP** 进行消息循环处理。

我们用 **IUP** 编写的图形界面程序的结构基本和上述代码类似。

在 Lua 形式下，我在导入 *iuplua* 模块后创建对话框。我们使用 Lua 的 *table* 来构建标签和对话框。这样做就不需要使用 *iup.SetAttribute* 函数。在 Lua 形式下，控件就是带有 **属性**和 **回调**的 *table*。在本例中，我们在控件创建时定义了它的属性。我们可以在这之后通过 *label.title* 或 *dlg.title* 修改它们的属性。在 Lua 形式下，许多函数有许多完全等价的调用形式，比如 *iup.ShowXY(dlg,iup.CENTER,iup.CENTER)* 和 *dlg:showxy(iup.CENTER,iup.CENTER)* 完全相同。

2.3 添加交互

在这一节，我们介绍如何给使用 **IUP** 编写的程序添加交互。我们为程序添加一个用于交互的 **按钮**。

C语言

```
#include <stdlib.h>
#include <iup.h>

int btn_exit_cb( Ihandle *self )
{
  IupMessage("Hello World Message", "Hello world from IUP.");
  /* Exits the main loop */
  return IUP_CLOSE;
}

int main(int argc, char **argv)
{
  Ihandle *dlg, *button, *vbox;

  IupOpen(&argc, &argv);

  button = IupButton("OK", NULL);
  vbox = IupVbox(
    button,
    NULL);
  dlg = IupDialog(vbox);
  IupSetAttribute(dlg, "TITLE", "Hello World 3");

  /* Registers callbacks */
  IupSetCallback(button, "ACTION", (Icallback) btn_exit_cb);
```

```
  IupShowXY(dlg, IUP_CENTER, IUP_CENTER);

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")

function btn_exit_cb(self)
  iup.Message("Hello World Message","Hello world from IUP.")
  -- Exits the main loop
  return iup.CLOSE
end

button = iup.button{title = "OK", action = btn_exit_cb}

vbox = iup.vbox{button}
dlg = iup.dialog{
  vbox,
  title = "Hello World 3"
}

dlg:showxy(iup.CENTER,iup.CENTER)

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

注意到，我们添加了一个叫做 *btn_exit_cb* 的函数，在这个函数里，我们显示了一条消息。最后它返回了 *IUP_CLOSE* 来退出消息循环。

我们使用**按钮**控件替换了上面的**标签**。*IupButton* 函数的第一个参数是按钮的**标签**，第二个参数目前已经被废弃，我们设置它为 *NULL* 即可。接着我们创建了一个垂直盒子，它被传递给 *IupDialog* 函数用作布局。

回调是在事件发生时，**IUP** 调用进行事件处理的函数。我们需要使用 *IupSetCallback* 函数设置相应事件的**回调**，**IUP** 才能在事件发生时调用它们。本例中，*btn_exit_cb* 函数在按钮按下后被调用。*IupSetCallback* 函数的第一个参数是 *Ihandle **，第二个参数是回调的名称，和属性一样，回调的名称都是大写，第三个参数是回调函数的地址。

当我们运行程序，会显示一个对话框。我们按下按钮后，会显示一个消息对话框，之后程序结束。虽然，例子很简单，但已经充分显示了如何进行控件的交互，其它控件的使用也只是换汤不换药。

在 Lua 形式下，我们使用 *table* 设置按钮的回调，所以代码中没有出现 *IupSetCallback* 函数。

2.4 添加布局元素

接下来，我们讨论布局的问题，在 **IUP** 中有主要的布局元素有: 垂直盒子 (*vbox*)，水平盒子 *hbox* 和填充 (*fill*)。除此之外还有 *zbox* 用于分层控件，只有被激活的层的控件才会被显示。

下面，我们添加一个标签到代码中，演示控件的布局。

C语言

```
#include <stdlib.h>
#include <iup.h>

int btn_exit_cb( Ihandle *self )
{
  /* Exits the main loop */
  return IUP_CLOSE;
}

int main(int argc, char **argv)
{
  Ihandle *dlg, *button, *label, *vbox;

  IupOpen(&argc, &argv);

  label = IupLabel("Hello world from IUP.");
  button = IupButton("OK", NULL);
  vbox = IupVbox(
    label,
    button,
    NULL);
  dlg = IupDialog(vbox);
  IupSetAttribute(dlg, "TITLE", "Hello World 4");

  /* Registers callbacks */
  IupSetCallback(button, "ACTION", (Icallback) btn_exit_cb);

  IupShowXY(dlg, IUP_CENTER, IUP_CENTER);

  IupMainLoop();

  IupClose();
```

```
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")

label = iup.label{title = "Hello world from IUP."}
button = iup.button{title = "OK"}

function button:action()
  -- Exits the main loop
  return iup.CLOSE
end

vbox = iup.vbox{label,button}
dlg = iup.dialog{
  vbox,
  title = "Hello World 4"
}

dlg:showxy(iup.CENTER,iup.CENTER)

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

我们可以看到**标签**和**按钮**控件被垂直排列。我们可以把 *vbox* 改为 *hbox* 看看会有什么变化。

在 Lua 形式下，我们可以通过使用 *button:action* 语法糖来简化回调函数定义，需要注意的是我们需要在 *button* 被构建后才能定义它的回调。

2.5 改善布局

我们已经理解了布局的基本概念，下面我们介绍**垂直盒子**和 **水平盒子**常用的三个属性: *ALIGNMENT*，*GAP* 和 *MARGIN*。

ALIGNMENT 定义了元素在盒子中如何分布。如果我们使用**垂直盒子**，它会控制水平布局，使用**水平盒子**，它会控制垂直布局。对于水平布局，它的值可以是 *ALEFT*，*ACENTER*，*ARIGHT*，对于垂直布局，它的值可以是 *ATOP*，*ACENTER*，*ABOTTOM*。它的缺省值是 *ALEFT* 和 *ATOP*。

GAP 定义了盒子中不同元素之间的空间距离，这个距离以像素为单位. 如果我们使用**垂直盒子**，它指的就是垂直空间距离. 使用**水平盒子**，它指的就是水平空间距离. 它的缺省值是 0[4]。

[4]表示元素之间没有多余的空间。

MARGIN 定义元素的边界大小，单位为像素。它的值的格式是 *width*×*height*，*width* 代表水平边界大小，*height* 代表垂直边界大小。它的缺省值是 0×0[5]。

接下来让我们演示如何使用这些**属性**。

Lua 形式的代码改变很少，我们仅仅添加了一个标签，并在之后设置了一些盒子布局的**属性**。

C语言

```
#include <stdlib.h>
#include <iup.h>

int btn_exit_cb( Ihandle *self )
{
  /* Exits the main loop */
  return IUP_CLOSE;
}

int main(int argc, char **argv)
{
  Ihandle *dlg, *button, *label, *vbox;

  IupOpen(&argc, &argv);

  label = IupLabel("Hello world from IUP.");
  button = IupButton("OK", NULL);
  vbox = IupVbox(
    label,
    button,
    NULL);
  IupSetAttribute(vbox, "ALIGNMENT", "ACENTER");
  IupSetAttribute(vbox, "GAP", "10");
  IupSetAttribute(vbox, "MARGIN", "10x10");

  dlg = IupDialog(vbox);
  IupSetAttribute(dlg, "TITLE", "Hello World 5");

  /* Registers callbacks */
  IupSetCallback(button, "ACTION", (Icallback) btn_exit_cb);

  IupShowXY(dlg, IUP_CENTER, IUP_CENTER);

  IupMainLoop();

  IupClose();
```

[5]表示没有边界。

```
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")

label = iup.label{title = "Hello world from IUP."}
button = iup.button{title = "OK"}

function button:action()
  -- Exits the main loop
  return iup.CLOSE
end

vbox = iup.vbox{
  label,
  button,
  alignment = "acenter",
  gap = "10",
  margin = "10x10"
}
dlg = iup.dialog{
  vbox,
  title = "Hello World 5"
}

dlg:showxy(iup.CENTER,iup.CENTER)

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

在水平盒子创建之后，我们添加了三行代码设置属性。我们可以对比发现它与第一个例子的消息对话框有很大不同。那么怎样设置这些属性才能使它的外观更接近在第一个例子中出现的消息对话框？交给大家思考。

3 简易记事本

3.1 主对话框

到现在为止我们已经认识了**标签**和**按钮**两个 **控件**。**标签**被用来显示文本和图像，通常不被用来进行交互操作。**按钮**允许用户触发点击事件。但是它们都不能接受用户输入数据。而 *IupText* 控

件则允许用户向其中输入文本。它有很多特有的**属性**。比如:*MULTILINE* 属性可以使 *IupText* 控件支持多行文本，通过这个控件我们可以构建一个简单的记事本程序。

让我们开始这项工作吧。

C语言

```
#include <stdlib.h>
#include <iup.h>

int main(int argc, char **argv)
{
  Ihandle *dlg, *multitext, *vbox;

  IupOpen(&argc, &argv);

  multitext = IupText(NULL);
  vbox = IupVbox(
    multitext,
    NULL);
  IupSetAttribute(multitext, "MULTILINE", "YES");
  IupSetAttribute(multitext, "EXPAND", "YES");

  dlg = IupDialog(vbox);
  IupSetAttribute(dlg, "TITLE", "Simple Notepad");
  IupSetAttribute(dlg, "SIZE", "QUARTERxQUARTER");

  IupShowXY(dlg, IUP_CENTER, IUP_CENTER);
  IupSetAttribute(dlg, "USERSIZE", NULL);

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")

multitext = iup.text{
  multiline = "YES",
  expand = "YES"
}
vbox = iup.vbox{
  multitext
```

```
}

dlg = iup.dialog{
  vbox,
  title = "Simple Notepad",
  size = "QUARTERxQUARTER"
}

dlg:showxy(iup.CENTER, iup.CENTER)
dlg.usersize = nil

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

我们在代码中设置 *IupText* 控件的 *MULTILINE* 属性的值为 *YES*，它的缺省值是 *NO*，可以尝试注释掉这一行，然后运行代码观察会有什么变化。

在代码中，我们设置了对话框的 *SIZE* 属性。由于 *IupText* 控件不会因为它的内容而改变控件大小，所以我们需要为对话框设置初始大小，避免出现非常小的对话框。在代码中，我们设置对话框的大小为 1/4 屏幕大小。*SIZE* 属性也被作为窗口最小的大小，可以在窗口显示之后设置 *USERSIZE* 属性来避免这个限制。可以尝试注释此行，观察对话框窗口大小改变的行为有什么不同。

短短几十行代码我们就创建了一个可以输入大量文本的编辑器。通常，我们希望我们的程序可以保存我们输入的信息，在下面的章节，我们会介绍如何实现这些功能。

3.2 添加菜单

几乎所有的程序都提供了一个菜单供用户来载入文件，保存文件，使用剪贴板以及进行其它一些数据处理操作。**IUP** 提供了四个接口元素给程序来操作菜单：*IupItem*，*IupMenu*，*IupSeparator* 和 *IupSubmenu*。

IupItem 表示一个菜单项目。当它被选中时，就会产生一个动作。

IupSeparator 在两个菜单项目之间创建一个分割线。它通常被用来对菜单项目进行分组。

IupSubmenu 用于创建子菜单，当它被选中时会打开子菜单。

IupMenu 用于存放上述三种类型的菜单元素。其它类型的元素放入 *IupMenu* 会引发错误。

接下来让我们为程序添加菜单。

C语言

```
#include <stdlib.h>
#include <iup.h>

int exit_cb(void)
{
```

```
  return IUP_CLOSE;
}

int main(int argc, char **argv)
{
  Ihandle *dlg, *multitext, *vbox;
  Ihandle *file_menu, *item_exit, *item_open, *item_saveas;
  Ihandle *sub1_menu, *menu;

  IupOpen(&argc, &argv);

  multitext = IupText(NULL);
  IupSetAttribute(multitext, "MULTILINE", "YES");
  IupSetAttribute(multitext, "EXPAND", "YES");

  item_open = IupItem("Open", NULL);
  item_saveas = IupItem("Save As", NULL);
  item_exit = IupItem("Exit", NULL);
  IupSetCallback(item_exit, "ACTION", (Icallback)exit_cb);

  file_menu = IupMenu(
    item_open,
    item_saveas,
    IupSeparator(),
    item_exit,
    NULL);

  sub1_menu = IupSubmenu("File", file_menu);

  menu = IupMenu(sub1_menu, NULL);

  vbox = IupVbox(
    multitext,
    NULL);

  dlg = IupDialog(vbox);
  IupSetAttributeHandle(dlg, "MENU", menu);
  IupSetAttribute(dlg, "TITLE", "Simple Notepad");
  IupSetAttribute(dlg, "SIZE", "QUARTERxQUARTER");

  IupShowXY(dlg, IUP_CENTER, IUP_CENTER);
  IupSetAttribute(dlg, "USERSIZE", NULL);
```

```
  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")

multitext = iup.text{
  multiline = "YES",
  expand = "YES"
}

item_open = iup.item{title="Open"}
item_save = iup.item{title="Save"}
item_exit = iup.item{title="Exit"}

function item_exit:action()
  return iup.CLOSE
end

file_menu = iup.menu{item_open,item_save,iup.separator{},item_exit}
sub1_menu = iup.submenu{file_menu, title = "File"}
menu = iup.menu{sub1_menu}

vbox = iup.vbox{
  multitext
}

dlg = iup.dialog{
  vbox,
  title = "Simple Notepad",
  size = "QUARTERxQUARTER",
  menu = menu
}

dlg:showxy(iup.CENTER,iup.CENTER)
dlg.usersize = nil

-- to be able to run this script inside another context
```

```
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

我们添加了一些菜单元素。可以看到我们设置 *item_exit* 的回调函数为 *exit_cb*。接着，我们使用 *IupMenu* 函数创建了一个叫做 *file_menu* 的菜单。菜单项目在窗口中出现的顺序和我们传参给 *IupMenu* 函数的顺序相同。也就是说 *item_open* 菜单项目会出现在 *item_save* 项目之上，以此类推。我们使用 *IupSeparator* 创建一个分割线来分割菜单。下一行，我们创建一个子菜单存储所有的菜单项目。为什么不直接使用 *file_menu* 呢？当然可以，但是这样它就会被作为主菜单，通常这不是我们希望的，我们可能更希望一个拥有文件，搜索，帮助和其它菜单的主菜单。

最后，我们设置对话框的 *MENU* 属性的值为我们创建的主菜单。这个设置不能使用 *IupSetAttribute* 函数[6]，而是使用 *IupSetAttributeHandle* 函数。

我们为 *exit* 菜单项设置了回调，但 *Open* 和 *Save* 菜单项目还没有设置，它们需要用到在接下来的章节介绍的其它 **IUP** 特性。

3.3 使用预定义对话框

在上一节中，我们添加了文件打开和文件保存的菜单项，但是没有给它们关联回调函数。原因是我们要使用预定义对话框来获取文件打开和保存的一些信息。

有许多通用的对话框比如文件选择对话框，字体选择对话框，颜色选择对话框等等。**IUP** 提供了这些通用对话框，不需要我们自己去构建它们。

现在我们更新我们的例子来使用预定义对话框来响应菜单项目。

C语言

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

/* global variable - to be used inside the menu callbacks */
Ihandle* multitext = NULL;

char* read_file(const char* filename)
{
  int size;
  char* str;
  FILE* file = fopen(filename, "rb");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return NULL;
  }
```

[6]因为这个属性的值是菜单句柄，不能使用普通的属性设置函数。

```
  /* calculate file size */
  fseek(file, 0, SEEK_END);
  size = ftell(file);
  fseek(file, 0, SEEK_SET);

  /* allocate memory for the file contents + nul terminator */
  str = malloc(size + 1);
  /* read all data at once */
  fread(str, size, 1, file);
  /* set the nul terminator */
  str[size] = 0;

  if (ferror(file))
    IupMessagef("Error", "Fail when reading from file: %s", filename);

  fclose(file);
  return str;
}

void write_file(const char* filename, const char* str, int count)
{
  FILE* file = fopen(filename, "w");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return;
  }

  fwrite(str, 1, count, file);

  if (ferror(file))
    IupMessagef("Error", "Fail when writing to file: %s", filename);

  fclose(file);
}

int open_cb(void)
{
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "OPEN");
  IupSetAttribute(filedlg, "EXTFILTER", "Text Files|*.txt|All Files|*.*|");
```

```
  IupPopup(filedlg, IUP_CENTER, IUP_CENTER);

  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    char* str = read_file(filename);
    if (str)
    {
      IupSetStrAttribute(multitext, "VALUE", str);
      free(str);
    }
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int saveas_cb(void)
{
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "SAVE");
  IupSetAttribute(filedlg, "EXTFILTER", "Text Files|*.txt|All Files|*.*|");

  IupPopup(filedlg, IUP_CENTER, IUP_CENTER);

  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    char* str = IupGetAttribute(multitext, "VALUE");
    int count = IupGetInt(multitext, "COUNT");
    write_file(filename, str, count);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int font_cb(void)
{
  Ihandle* fontdlg = IupFontDlg();
  char* font = IupGetAttribute(multitext, "FONT");
  IupSetStrAttribute(fontdlg, "VALUE", font);
```

```
  IupPopup(fontdlg, IUP_CENTER, IUP_CENTER);

  if (IupGetInt(fontdlg, "STATUS") == 1)
  {
    char* font = IupGetAttribute(fontdlg, "VALUE");
    IupSetStrAttribute(multitext, "FONT", font);
  }

  IupDestroy(fontdlg);
  return IUP_DEFAULT;
}

int about_cb(void)
{
  IupMessage("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣S
  return IUP_DEFAULT;
}

int exit_cb(void)
{
  return IUP_CLOSE;
}

int main(int argc, char **argv)
{
  Ihandle *dlg, *vbox;
  Ihandle *file_menu, *item_exit, *item_open, *item_saveas;
  Ihandle *format_menu, *item_font;
  Ihandle *help_menu, *item_about;
  Ihandle *sub_menu_file, *sub_menu_format, *sub_menu_help, *menu;

  IupOpen(&argc, &argv);

  multitext = IupText(NULL);
  IupSetAttribute(multitext, "MULTILINE", "YES");
  IupSetAttribute(multitext, "EXPAND", "YES");

  item_open = IupItem("Open...", NULL);
  item_saveas = IupItem("Save␣As...", NULL);
  item_exit = IupItem("Exit", NULL);
  item_font = IupItem("Font...", NULL);
  item_about = IupItem("About...", NULL);
```

```
IupSetCallback(item_exit, "ACTION", (Icallback)exit_cb);
IupSetCallback(item_open, "ACTION", (Icallback)open_cb);
IupSetCallback(item_saveas, "ACTION", (Icallback)saveas_cb);
IupSetCallback(item_font, "ACTION", (Icallback)font_cb);
IupSetCallback(item_about, "ACTION", (Icallback)about_cb);

file_menu = IupMenu(
  item_open,
  item_saveas,
  IupSeparator(),
  item_exit,
  NULL);
format_menu = IupMenu(
  item_font,
  NULL);
help_menu = IupMenu(
  item_about,
  NULL);

sub_menu_file = IupSubmenu("File", file_menu);
sub_menu_format = IupSubmenu("Format", format_menu);
sub_menu_help = IupSubmenu("Help", help_menu);

menu = IupMenu(
  sub_menu_file,
  sub_menu_format,
  sub_menu_help,
  NULL);

vbox = IupVbox(
  multitext,
  NULL);

dlg = IupDialog(vbox);
IupSetAttributeHandle(dlg, "MENU", menu);
IupSetAttribute(dlg, "TITLE", "Simple Notepad");
IupSetAttribute(dlg, "SIZE", "QUARTERxQUARTER");

IupShowXY(dlg, IUP_CENTER, IUP_CENTER);
IupSetAttribute(dlg, "USERSIZE", NULL);
```

```
  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")

function read_file(filename)
  local ifile = io.open(filename, "r")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return nil
  end

  local str = ifile:read("*a")
  if (not str) then
    iup.Message("Error", "Fail when reading from file: " .. filename)
    return nil
  end

  ifile:close()
  return str
end

function write_file(filename, str)
  local ifile = io.open(filename, "w")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return
  end

  if (not ifile:write(str)) then
    iup.Message("Error", "Fail when writing to file: " .. filename)
  end

  ifile:close()
end

multitext = iup.text{
  multiline = "YES",
```

```
  expand = "YES"
}

item_open = iup.item{title="Open..."}
item_saveas = iup.item{title="Save As..."}
item_font = iup.item{title="Font..."}
item_about = iup.item{title="About..."}
item_exit = iup.item{title="Exit"}

function item_open:action()
  local filedlg = iup.filedlg{
    dialogtype = "OPEN",
    filter = "*.txt",
    filterinfo = "Text Files",
    }

  filedlg:popup(iup.CENTER, iup.CENTER)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    local str = read_file(filename)
    if (str) then
      multitext.value = str
    end
  end
  filedlg:destroy()
end

function item_saveas:action()
  local filedlg = iup.filedlg{
    dialogtype = "SAVE",
    filter = "*.txt",
    filterinfo = "Text Files",
    }

  filedlg:popup(iup.CENTER, iup.CENTER)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    write_file(filename, multitext.value)
  end
  filedlg:destroy()
```

```
end

function item_font:action()
  local font = multitext.font
  local fontdlg = iup.fontdlg{value = font}

  fontdlg:popup(iup.CENTER, iup.CENTER)

  if (tonumber(fontdlg.status) == 1) then
    multitext.font = fontdlg.value
  end

  fontdlg:destroy()
end

function item_about:action()
  iup.Message("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣
end

function item_exit:action()
  return iup.CLOSE
end

file_menu = iup.menu{item_open,item_saveas,iup.separator{},item_exit}
format_menu = iup.menu{item_font}
help_menu = iup.menu{item_about}
sub_menu_file = iup.submenu{file_menu, title = "File"}
sub_menu_format = iup.submenu{format_menu, title = "Format"}
sub_menu_help = iup.submenu{help_menu, title = "Help"}

menu = iup.menu{
  sub_menu_file,
  sub_menu_format,
  sub_menu_help
  }

vbox = iup.vbox{
  multitext
}

dlg = iup.dialog{
  vbox,
```

```
  title = "Simple Notepad",
  size = "QUARTERxQUARTER",
  menu = menu
}

dlg:showxy(iup.CENTER, iup.CENTER)
dlg.usersize = nil

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

我们需要在菜单回调中访问文本框控件。有许多种方法做到。最简单的就是定义文本框控件为全局变量。我们在本例中暂时这样做，但这种方法并不被推荐使用。在下面的例子中，我们将会介绍不使用全局变量的方法。

现在我们先把注意力放在新出现的函数上。首先是 *open_cb* 函数。当用户点击打开文件菜单项目时，该函数被调用。在这个函数中我们调用 *IupFileDlg* 函数来使用预定义的文件对话框。我们设置它的 *DIALOGTYPE* 属性为 *open*，然后设置 *EXTFILTER* 属性为 *Text Files|*.txt|All Files|*.*|*，这个属性会根据后缀过滤掉我们不需要的文件类型。

然后，我们调用 *IupPopup* 函数，它和 *IupShow* 函数类似，但是它限制用户只能与这个对话框进行交互。在许多平台它就是一个模态对话框。它的参数 *IupShow* 相同。

然后，我们调用 *IupGetInt* 函数获取对话框的结果信息。有人会问为什么不使用 *IupGetAttribute* 函数？原因是 *IupGetAttribute* 函数返回的是字符串，但我们在这里的结果则是一个整型数，所以用 *IupGetInt* 函数更合适。

我们接受的结果显示用户作出了合法的操作后，我们就可以使用 *IupGetAttribute* 函数来获取被选择的文件信息。只要查询它的 *VALUE* 属性即可。接着，我们读取文件，并把读到的文件内容设置成文本框的 *VALUE* 属性值。这个操作可以通过 *IupSetStrAttribute* 函数进行。这个函数会对文本进行复制，所以在调用这个函数之后我们可以释放掉作为参数的字符串。

这个对话框的任务已经结束了，我们可以使用 *IupDestroy* 函数来销毁它。

我们看到，还有其它一些回调和预定义对话框的使用。但这些内容都是类似的。

在 *about_cb* 回调中，我们使用 *IupMessage* 函数显示文本给用户。

接下来并没有什么新的东西出现，除了我们在有对话框的菜单项目的文本描述中添加了... 表示有对话框弹出[7]。

现在，我们有了一个使用 **IUP** 库编写的文本编辑器。目前，我们大量使用了预定义的对话框，如果我们需要使用一些自定义的对话框又要如何做呢？下一节，我们将会介绍。

3.4 自定义对话框

在前一节，我们了解到了 **IUP** 提供的预定义对话框，使用它们可以节约大量的开发时间。但有时预定义的对话框可能满足不了我们的要求，需要我们自己去构建自己的对话框。就像我们在第一个例子中做的那样。现在我们需要了解如何同时处理超过一个对话框。

[7]这是一个 GUI 规范

我们在 *Edit* 菜单下添加 *Find* 和 *Go To* 两个菜单项目。*Find* 菜单项用于搜索并高亮在文本框中的指定文本。*Go TO* 菜单项用于跳转输入光标到文本的指定行。

C语言

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <iup.h>


/********************************* Utilities *************************************


int str_compare(const char *l, const char *r, int casesensitive)
{
  if (!l || !r)
    return 0;

  while (*l && *r)
  {
    int diff;
    char l_char = *l,
      r_char = *r;

    /* compute the difference of both characters */
    if (casesensitive)
      diff = l_char - r_char;
    else
      diff = tolower((int)l_char) - tolower((int)r_char);

    /* if they differ we have a result */
    if (diff != 0)
      return 0;

    /* otherwise process the next characters */
    ++l;
    ++r;
  }

  /* check also for terminator */
  if (*l == *r)
    return 1;
```

```
  if (*r == 0)
    return 1;  /* if second string is at terminator, then it is partially equal */

  return 0;
}

int str_find(const char *str, const char *str_to_find, int casesensitive)
{
  int i, str_len, str_to_find_len, count;

  if (!str || str[0] == 0 || !str_to_find || str_to_find[0] == 0)
    return -1;

  str_len = (int)strlen(str);
  str_to_find_len = (int)strlen(str_to_find);
  count = str_len - str_to_find_len;
  if (count < 0)
    return -1;

  count++;

  for (i = 0; i<count; i++)
  {
    if (str_compare(str, str_to_find, casesensitive))
      return i;

    str++;
  }

  return -1;
}

char* read_file(const char* filename)
{
  int size;
  char* str;
  FILE* file = fopen(filename, "rb");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return NULL;
```

```
  }

  /* calculate file size */
  fseek(file, 0, SEEK_END);
  size = ftell(file);
  fseek(file, 0, SEEK_SET);

  /* allocate memory for the file contents + nul terminator */
  str = malloc(size + 1);
  /* read all data at once */
  fread(str, size, 1, file);
  /* set the nul terminator */
  str[size] = 0;

  if (ferror(file))
    IupMessagef("Error", "Fail when reading from file: %s", filename);

  fclose(file);
  return str;
}

void write_file(const char* filename, const char* str, int count)
{
  FILE* file = fopen(filename, "w");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return;
  }

  fwrite(str, 1, count, file);

  if (ferror(file))
    IupMessagef("Error", "Fail when writing to file: %s", filename);

  fclose(file);
}


/********************************* Callbacks *************************************
```

```
int item_open_action_cb(Ihandle* item_open)
{
  Ihandle* multitext = IupGetDialogChild(item_open, "MULTITEXT");
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "OPEN");
  IupSetAttribute(filedlg, "EXTFILTER", "Text␣Files|*.txt|All␣Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_open));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    char* str = read_file(filename);
    if (str)
    {
      IupSetStrAttribute(multitext, "VALUE", str);
      free(str);
    }
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_saveas_action_cb(Ihandle* item_saveas)
{
  Ihandle* multitext = IupGetDialogChild(item_saveas, "MULTITEXT");
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "SAVE");
  IupSetAttribute(filedlg, "EXTFILTER", "Text␣Files|*.txt|All␣Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_saveas));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    char* str = IupGetAttribute(multitext, "VALUE");
    int count = IupGetInt(multitext, "COUNT");
    write_file(filename, str, count);
  }
```

```
  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_exit_action_cb(void)
{
  return IUP_CLOSE;
}

int goto_ok_action_cb(Ihandle* bt_ok)
{
  int line_count = IupGetInt(bt_ok, "TEXT_LINECOUNT");
  Ihandle* txt = IupGetDialogChild(bt_ok, "LINE_TEXT");
  int line = IupGetInt(txt, "VALUE");
  if (line < 1 || line >= line_count)
  {
    IupMessage("Error", "Invalid line number.");
    return IUP_DEFAULT;
  }

  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "1");
  return IUP_CLOSE;
}

int goto_cancel_action_cb(Ihandle* bt_ok)
{
  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "0");
  return IUP_CLOSE;
}

int item_goto_action_cb(Ihandle* item_goto)
{
  Ihandle* multitext = IupGetDialogChild(item_goto, "MULTITEXT");
  Ihandle *dlg, *box, *bt_ok, *bt_cancel, *txt, *lbl;

  int line_count = IupGetInt(multitext, "LINECOUNT");

  lbl = IupLabel(NULL);
  IupSetfAttribute(lbl, "TITLE", "Line Number [1-%d]:", line_count);
  txt = IupText(NULL);
  IupSetAttribute(txt, "MASK", IUP_MASK_UINT);  /* unsigned integer numbers only */
```

```
  IupSetAttribute(txt, "NAME", "LINE_TEXT");
  IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
  bt_ok = IupButton("OK", NULL);
  IupSetInt(bt_ok, "TEXT_LINECOUNT", line_count);
  IupSetAttribute(bt_ok, "PADDING", "10x2");
  IupSetCallback(bt_ok, "ACTION", (Icallback)goto_ok_action_cb);
  bt_cancel = IupButton("Cancel", NULL);
  IupSetCallback(bt_cancel, "ACTION", (Icallback)goto_cancel_action_cb);
  IupSetAttribute(bt_cancel, "PADDING", "10x2");

  box = IupVbox(
    lbl,
    txt,
    IupSetAttributes(IupHbox(
      IupFill(),
      bt_ok,
      bt_cancel,
      NULL), "NORMALIZESIZE=HORIZONTAL"),
    NULL);
  IupSetAttribute(box, "MARGIN", "10x10");
  IupSetAttribute(box, "GAP", "5");

  dlg = IupDialog(box);
  IupSetAttribute(dlg, "TITLE", "Go To Line");
  IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
  IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_ok);
  IupSetAttributeHandle(dlg, "DEFAULTESC", bt_cancel);
  IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_goto));

  IupPopup(dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(dlg, "STATUS") == 1)
  {
    int line = IupGetInt(txt, "VALUE");
    int pos;
    IupTextConvertLinColToPos(multitext, line, 0, &pos);
    IupSetInt(multitext, "CARETPOS", pos);
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }

  IupDestroy(dlg);
```

```
  return IUP_DEFAULT;
}

int find_next_action_cb(Ihandle* bt_next)
{
  Ihandle* multitext = (Ihandle*)IupGetAttribute(bt_next, "MULTITEXT");
  char* str = IupGetAttribute(multitext, "VALUE");
  int find_pos = IupGetInt(multitext, "FIND_POS");

  Ihandle* txt = IupGetDialogChild(bt_next, "FIND_TEXT");
  char* str_to_find = IupGetAttribute(txt, "VALUE");

  Ihandle* find_case = IupGetDialogChild(bt_next, "FIND_CASE");
  int casesensitive = IupGetInt(find_case, "VALUE");

  int pos = str_find(str + find_pos, str_to_find, casesensitive);
  if (pos >= 0)
    pos += find_pos;
  else if (find_pos > 0)
    pos = str_find(str, str_to_find, casesensitive);  /* try again from the start *

  if (pos >= 0)
  {
    int lin, col,
      end_pos = pos + (int)strlen(str_to_find);

    IupSetInt(multitext, "FIND_POS", end_pos);

    IupSetFocus(multitext);
    IupSetfAttribute(multitext, "SELECTIONPOS", "%d:%d", pos, end_pos);

    IupTextConvertPosToLinCol(multitext, pos, &lin, &col);
    IupTextConvertLinColToPos(multitext, lin, 0, &pos);  /* position at col=0, just
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }
  else
    IupMessage("Warning", "Text not found.");

  return IUP_DEFAULT;
}

int find_close_action_cb(Ihandle* bt_close)
```

```
{
  IupHide(IupGetDialog(bt_close));
  return IUP_DEFAULT;
}

int item_find_action_cb(Ihandle* item_find)
{
  Ihandle* dlg = (Ihandle*)IupGetAttribute(item_find, "FIND_DIALOG");
  if (!dlg)
  {
    Ihandle* multitext = IupGetDialogChild(item_find, "MULTITEXT");
    Ihandle *box, *bt_next, *bt_close, *txt, *find_case;

    txt = IupText(NULL);
    IupSetAttribute(txt, "NAME", "FIND_TEXT");
    IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
    find_case = IupToggle("Case Sensitive", NULL);
    IupSetAttribute(find_case, "NAME", "FIND_CASE");
    bt_next = IupButton("Find Next", NULL);
    IupSetAttribute(bt_next, "PADDING", "10x2");
    IupSetCallback(bt_next, "ACTION", (Icallback)find_next_action_cb);
    bt_close = IupButton("Close", NULL);
    IupSetCallback(bt_close, "ACTION", (Icallback)find_close_action_cb);
    IupSetAttribute(bt_close, "PADDING", "10x2");

    box = IupVbox(
      IupLabel("Find What:"),
      txt,
      find_case,
      IupSetAttributes(IupHbox(
        IupFill(),
        bt_next,
        bt_close,
        NULL), "NORMALIZESIZE=HORIZONTAL"),
      NULL);
    IupSetAttribute(box, "MARGIN", "10x10");
    IupSetAttribute(box, "GAP", "5");

    dlg = IupDialog(box);
    IupSetAttribute(dlg, "TITLE", "Find");
    IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
    IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_next);
```

```
    IupSetAttributeHandle(dlg, "DEFAULTESC", bt_close);
    IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_find));

    /* Save the multiline to acess it from the callbacks */
    IupSetAttribute(dlg, "MULTITEXT", (char*)multitext);

    /* Save the dialog to reuse it */
    IupSetAttribute(item_find, "FIND_DIALOG", (char*)dlg);
  }

  /* centerparent first time, next time reuse the last position */
  IupShowXY(dlg, IUP_CURRENT, IUP_CURRENT);

  return IUP_DEFAULT;
}

int item_font_action_cb(Ihandle* item_font)
{
  Ihandle* multitext = IupGetDialogChild(item_font, "MULTITEXT");
  Ihandle* fontdlg = IupFontDlg();
  char* font = IupGetAttribute(multitext, "FONT");
  IupSetStrAttribute(fontdlg, "VALUE", font);
  IupSetAttributeHandle(fontdlg, "PARENTDIALOG", IupGetDialog(item_font));

  IupPopup(fontdlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(fontdlg, "STATUS") == 1)
  {
    font = IupGetAttribute(fontdlg, "VALUE");
    IupSetStrAttribute(multitext, "FONT", font);
  }

  IupDestroy(fontdlg);
  return IUP_DEFAULT;
}

int item_about_action_cb(void)
{
  IupMessage("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣S
  return IUP_DEFAULT;
}
```

```
/********************************** Main *****************************************/


int main(int argc, char **argv)
{
  Ihandle *dlg, *vbox, *multitext, *menu;
  Ihandle *sub_menu_file, *file_menu, *item_exit, *item_open, *item_saveas;
  Ihandle *sub_menu_edit, *edit_menu, *item_find, *item_goto;
  Ihandle *sub_menu_format, *format_menu, *item_font;
  Ihandle *sub_menu_help, *help_menu, *item_about;

  IupOpen(&argc, &argv);

  multitext = IupText(NULL);
  IupSetAttribute(multitext, "MULTILINE", "YES");
  IupSetAttribute(multitext, "EXPAND", "YES");
  IupSetAttribute(multitext, "NAME", "MULTITEXT");

  item_open = IupItem("Open...", NULL);
  item_saveas = IupItem("Save As...", NULL);
  item_exit = IupItem("Exit", NULL);
  item_find = IupItem("Find..", NULL);
  item_goto = IupItem("Go To...", NULL);
  item_font = IupItem("Font...", NULL);
  item_about = IupItem("About...", NULL);

  IupSetCallback(item_open, "ACTION", (Icallback)item_open_action_cb);
  IupSetCallback(item_saveas, "ACTION", (Icallback)item_saveas_action_cb);
  IupSetCallback(item_exit, "ACTION", (Icallback)item_exit_action_cb);
  IupSetCallback(item_find, "ACTION", (Icallback)item_find_action_cb);
  IupSetCallback(item_goto, "ACTION", (Icallback)item_goto_action_cb);
  IupSetCallback(item_font, "ACTION", (Icallback)item_font_action_cb);
  IupSetCallback(item_about, "ACTION", (Icallback)item_about_action_cb);

  file_menu = IupMenu(
    item_open,
    item_saveas,
    IupSeparator(),
    item_exit,
    NULL);
  edit_menu = IupMenu(
```

```
    item_find,
    item_goto,
    NULL);
  format_menu = IupMenu(
    item_font,
    NULL);
  help_menu = IupMenu(
    item_about,
    NULL);

  sub_menu_file = IupSubmenu("File", file_menu);
  sub_menu_edit = IupSubmenu("Edit", edit_menu);
  sub_menu_format = IupSubmenu("Format", format_menu);
  sub_menu_help = IupSubmenu("Help", help_menu);

  menu = IupMenu(
    sub_menu_file,
    sub_menu_edit,
    sub_menu_format,
    sub_menu_help,
    NULL);

  vbox = IupVbox(
    multitext,
    NULL);

  dlg = IupDialog(vbox);
  IupSetAttributeHandle(dlg, "MENU", menu);
  IupSetAttribute(dlg, "TITLE", "Simple Notepad");
  IupSetAttribute(dlg, "SIZE", "HALFxHALF");

  /* parent for pre-defined dialogs in closed functions (IupMessage) */
  IupSetAttributeHandle(NULL, "PARENTDIALOG", dlg);

  IupShowXY(dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
  IupSetAttribute(dlg, "USERSIZE", NULL);

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")


--********************************* Utilities *********************************


function str_find(str, str_to_find, casesensitive, start)
  if (not casesensitive) then
    return str_find(string.lower(str), string.lower(str_to_find), true, start)
  end

  return string.find(str, str_to_find, start, true)
end

function read_file(filename)
  local ifile = io.open(filename, "r")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return nil
  end

  local str = ifile:read("*a")
  if (not str) then
    iup.Message("Error", "Fail when reading from file: " .. filename)
    return nil
  end

  ifile:close()
  return str
end

function write_file(filename, str)
  local ifile = io.open(filename, "w")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return
  end

  if (not ifile:write(str)) then
    iup.Message("Error", "Fail when writing to file: " .. filename)
```

```
  end

  ifile:close()
end


--********************************* Main (Part 1/2) ****************************


multitext = iup.text{
  multiline = "YES",
  expand = "YES"
}

item_open = iup.item{title="Open..."}
item_saveas = iup.item{title="Save As..."}
item_font = iup.item{title="Font..."}
item_about = iup.item{title="About..."}
item_find = iup.item{title="Find..."}
item_goto = iup.item{title="Go To..."}
item_exit = iup.item{title="Exit"}


--********************************* Callbacks **********************************


function item_open:action()
  local filedlg = iup.filedlg{
    dialogtype = "OPEN",
    filter = "*.txt",
    filterinfo = "Text Files",
    parentdialog=iup.GetDialog(self)
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    local str = read_file(filename)
    if (str) then
      multitext.value = str
    end
```

```
  end
  filedlg:destroy()
end

function item_saveas:action()
  local filedlg = iup.filedlg{
    dialogtype = "SAVE",
    filter = "*.txt",
    filterinfo = "Text Files",
    parentdialog=iup.GetDialog(self)
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    write_file(filename, multitext.value)
  end
  filedlg:destroy()
end

function item_exit:action()
  return iup.CLOSE
end

function item_goto:action()
  local line_count = multitext.linecount
  local lbl_goto = iup.label{title = "Line Number [1-"..line_count.."]:"}
  local txt_goto = iup.text{mask = iup.MASK_UINT, visiblecolumns = 20} --unsigned i

  local bt_goto_ok = iup.button{title = "OK", text_linecount = 0, padding = "10x2"}
  bt_goto_ok.text_linecount = line_count
  function bt_goto_ok:action()
    local line_count = tonumber(self.text_linecount)
    local line = tonumber(txt_goto.value)
    if (line < 1 or line >= line_count) then
      iup.Message("Error", "Invalid line number.")
      return
    end
    goto_dlg.status = 1
    return iup.CLOSE
  end
```

```
  local bt_goto_cancel = iup.button{title = "Cancel", padding = "10x2"}
  function bt_goto_cancel:action()
    goto_dlg.status = 0
    return iup.CLOSE
  end

  local box = iup.vbox{
    lbl_goto,
    txt_goto,
    iup.hbox{
      iup.fill{},
      bt_goto_ok,
      bt_goto_cancel,
      normalizesize="HORIZONTAL",
    },
    margin = "10x10",
    gap = "5",
  }
  goto_dlg = iup.dialog{
    box,
    title = "Go To Line",
    dialogframe = "Yes",
    defaultenter = bt_goto_ok,
    defaultesc = bt_goto_cancel,
    parentdialog = iup.GetDialog(self)
  }

  goto_dlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(goto_dlg.status) == 1) then
    local line = txt_goto.value
    local pos = iup.TextConvertLinColToPos(multitext, line, 0)
    multitext.caretpos = pos
    multitext.scrolltopos = pos
  end

  goto_dlg:destroy()
end

function item_find:action()
  local find_dlg = self.find_dialog
```

```
if (not find_dlg) then
  local find_txt = iup.text{visiblecolumns = "20"}
  local find_case = iup.toggle{title = "Case Sensitive"}
  local bt_find_next = iup.button{title = "Find Next", padding = "10x2"}
  local bt_find_close = iup.button{title = "Close", padding = "10x2"}

  function bt_find_next:action()
    local find_pos = tonumber(find_dlg.find_pos)
    local str_to_find = find_txt.value

    local casesensitive = (find_case.value == "ON")

    -- test again, because it can be called from the hot key
    if (not str_to_find or str_to_find:len()==0) then
      return
    end

    if (not find_pos) then
      find_pos = 1
    end

    local str = multitext.value

    local pos, end_pos = str_find(str, str_to_find, casesensitive, find_pos)
    if (not pos) then
      pos, end_pos = str_find(str, str_to_find, casesensitive, 1)  -- try again fr
    end

    if (pos) and (pos > 0) then
      pos = pos - 1
      find_dlg.find_pos = end_pos

      iup.SetFocus(multitext)
      multitext.selectionpos = pos..":"..end_pos

      local lin, col = iup.TextConvertPosToLinCol(multitext, pos)
      local pos = iup.TextConvertLinColToPos(multitext, lin, 0)  -- position at c
      multitext.scrolltopos = pos
    else
      find_dlg.find_pos = nil
      iup.Message("Warning", "Text not found.")
    end
```

```
    end

    function bt_find_close:action()
      iup.Hide(iup.GetDialog(self))  -- do not destroy, just hide
    end

    box = iup.vbox{
      iup.label{title = "Find What:"},
      find_txt,
      find_case,
      iup.hbox{
        iup.fill{},
        bt_find_next,
        bt_find_close,
        normalizesize="HORIZONTAL",
      },
      margin = "10x10",
      gap = "5",
    }

    find_dlg = iup.dialog{
      box,
      title = "Find",
      dialogframe = "Yes",
      defaultenter = bt_next,
      defaultesc = bt_close,
      parentdialog = iup.GetDialog(self)
      }

    -- Save the dialog to reuse it
    self.find_dialog = find_dlg -- from the main dialog */
  end

  -- centerparent first time, next time reuse the last position
  find_dlg:showxy(iup.CURRENT, iup.CURRENT)
end

function item_font:action()
  local font = multitext.font
  local fontdlg = iup.fontdlg{value = font, parentdialog=iup.GetDialog(self)}

  fontdlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)
```

```
  if (tonumber(fontdlg.status) == 1) then
    multitext.font = fontdlg.value
  end

  fontdlg:destroy()
end

function item_about:action()
  iup.Message("About", "   Simple Notepad\n\nAutors:\n   Gustavo Lyrio\n   Antonio 
end


--********************************* Main (Part 2/2) ****************************


file_menu = iup.menu{item_open,item_saveas,iup.separator{},item_exit}
edit_menu = iup.menu{item_find, item_goto}
format_menu = iup.menu{item_font}
help_menu = iup.menu{item_about}
sub_menu_file = iup.submenu{file_menu, title = "File"}
sub_menu_edit = iup.submenu{edit_menu, title = "Edit"}
sub_menu_format = iup.submenu{format_menu, title = "Format"}
sub_menu_help = iup.submenu{help_menu, title = "Help"}

menu = iup.menu{
  sub_menu_file,
  sub_menu_edit,
  sub_menu_format,
  sub_menu_help,
  }

vbox = iup.vbox{
  multitext
}

dlg = iup.dialog{
  vbox,
  title = "Simple Notepad",
  size = "HALFxHALF",
  menu = menu
}
```

```
-- parent for pre-defined dialogs in closed functions (IupMessage)
iup.SetGlobal("PARENTDIALOG", iup.SetHandleName(dlg))

dlg:showxy(iup.CENTER,iup.CENTER)
dlg.usersize = nil

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

在代码中可以看到我们添加了 *str_compare* 和 *str_find* 函数用于实现字符串的搜索。但这些代码不是我们的重点，在此略去解释。

我们发现部分函数名字在新的代码中被改变。我们把函数的名字命名地更加规范，这在大型程序中非常重要，可以减少维护的成本。

首先我们看 *item_find_action_cb* 函数。在这个函数中我们构建了字符串查找对话框。它包括这些部分：一个文本框用于接受需要查找的字符串，一个查找按钮，一个关闭按钮，和两个新的 **IUP** 元素:*IupToggle* 和 *IupFill*。

IupToggle 是一个具有开和关两种状态的按钮，它被选中时就会执行回调。通常,*Toggle* 用来设置标志。在这里我们用来让用户选择搜索是否是大小写相关的。

IupFill 是一个非常特殊的元素。就像它的名字表达的那样，它被用来在对话框中填充空白。换句话说，它可以用来布局 **IUP** 元素。把 *IupFille* 放置在 *IupHbox* 中，它会分割两个元素。*IupFill* 也具有 *SIZE* 属性，可以控制它占用的空间大小。我们可以通过经验设置合适的值给它。在本例中，我们使用 *IupFill* 来使 *Find* 按钮和 *Close* 按钮布局在水平盒子的右边。

注意到，我们的对话框有一些新的参数。*DIALOGFRAME* 会移除对话框的最小，最大按钮，并且它会重新设置对话框的大小。*DEFAULTENTER* 属性定义了用户按下回车键默认响应的按钮，在本例中，它的效果和按下 *next_bt* 按钮一样。*DEFAULTESC* 属性定义按下 ESC 键默认响应的按钮。*PARENTALDIALOG* 属性用于设置对话框的父窗口。通过 *IupGetDialog* 函数我们可以获取一个元素的父窗口。这允许我们在查找对话框对父窗口的控件进行操作。

在下面的两行，我们使用了两个自定义的属性存储程序指针。每一个 **IUP** 元素都可以拥有任意数量的自定义属性。我们可以用它来存储一些信息。在这里，我们使用了一个叫做 *MULTITEXT* 的属性来存储文本框控件的指针，这样我们就可以很方便地在回调函数中访问文本框控件。这也避免了使用全局属性。我们还为 *find_item* 元素创建了一个叫做 *FIND_DIALOG* 的属性来存储查找对话框指针，只有第一次打开查找对话框时才会创建它，再次访问时，我们就会通过这个属性来复用之前创建的对话框。

接着我们使用 *IupShowXY* 函数来显示对话框，并给它传递了 *IUP_CURRENT* 参数。第一次时，我们的查找对话框会被放置在父窗口的中央。下一次打开查找对话框，它会被放在上次关闭它的地方。

我们已经创建好了查找对话框，接下来是编写回调函数来进行字符串查找。

让我们来看下 *find_next_action_cb* 回调。这个回调实际执行了字符串查找的工作。回调中出现了一些新的函数。*IupGetDialogChild* 函数返回设置了指定 *NAME* 属性值的控件。在本例中，我

们使用它来获取文本框控件的句柄。我们可以通过它来避免使用自定义属性和全局变量。但需要注意的是，它只能在同一个对话框下工作。我们从控件获取要查找的文本以及是否大小写无关的信息后就可以执行搜索操作了。搜索后，如果得到有效的结果，就保存查找到的位置到自定义属性，然后调用 *IupSetFocus* 函数设置焦点到文本框控件。然后，我们使用 *IupTextConvertPosToLinCol* 和 *IupTextConvertLinColToPos* 函数来跳转光标并选中查找到的文本。

除了 *next_bt* 按钮，查找对话框还有 *close_bt* 按钮，它也有一个回调。*Find_close_action_cb* 回调用来关闭查找对话框。在这个回调中，我们使用了 *IupHide* 函数。它把对话框隐藏，但这并不是销毁，我们可以再次显示它。

Go To 对话框的工作机理和查找对话框是相同的。所以这里就不再解释它的代码。

3.5 添加工具栏和状态栏

现在我们已经知道了如何使用预定义的对话框和如何构建自定义的对话框。接下来，我们来学习使用**工具栏**和**状态栏**。

工具栏是一个按钮的集合，通常被放置在窗口的顶部，位于菜单之下。我们可以使用 *IupButton* 元素的 *IMAGE* 属性来构建带有图标的工具栏按钮。和预定义对话框一样，**IUP** 也提供一些预定义的图标供按钮使用。这些图像位于一个叫做 *IupImageLib* 的附加库中。使用这个库，需要在 *IupOpen* 函数调用后，调用 *IupImageLibOpen* 函数。

状态栏通常出现在窗口的底部，通常用来显示一些程序运行过程中的信息。我们会在**状态栏**中放置**标签**来显示文本信息。在本例中，我们会在**状态栏**显示光标的位置信息。我们通过 *IupText* 控件的 *CARET_CB* 回调来跟踪光标变化，更新标签就可以完成这个工作。

C语言

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <iup.h>


/********************************** Utilities *************************************


int str_compare(const char *l, const char *r, int casesensitive)
{
  if (!l || !r)
    return 0;

  while (*l && *r)
  {
    int diff;
    char l_char = *l,
      r_char = *r;
```

```
    /* compute the difference of both characters */
    if (casesensitive)
      diff = l_char - r_char;
    else
      diff = tolower((int)l_char) - tolower((int)r_char);

    /* if they differ we have a result */
    if (diff != 0)
      return 0;

    /* otherwise process the next characters */
    ++l;
    ++r;
  }

  /* check also for terminator */
  if (*l == *r)
    return 1;

  if (*r == 0)
    return 1;  /* if second string is at terminator, then it is partially equal */

  return 0;
}

int str_find(const char *str, const char *str_to_find, int casesensitive)
{
  int i, str_len, str_to_find_len, count;

  if (!str || str[0] == 0 || !str_to_find || str_to_find[0] == 0)
    return -1;

  str_len = (int)strlen(str);
  str_to_find_len = (int)strlen(str_to_find);
  count = str_len - str_to_find_len;
  if (count < 0)
    return -1;

  count++;

  for (i = 0; i<count; i++)
  {
```

```
    if (str_compare(str, str_to_find, casesensitive))
      return i;

    str++;
  }

  return -1;
}

char* read_file(const char* filename)
{
  int size;
  char* str;
  FILE* file = fopen(filename, "rb");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return NULL;
  }

  /* calculate file size */
  fseek(file, 0, SEEK_END);
  size = ftell(file);
  fseek(file, 0, SEEK_SET);

  /* allocate memory for the file contents + nul terminator */
  str = malloc(size + 1);
  /* read all data at once */
  fread(str, size, 1, file);
  /* set the nul terminator */
  str[size] = 0;

  if (ferror(file))
    IupMessagef("Error", "Fail when reading from file: %s", filename);

  fclose(file);
  return str;
}

void write_file(const char* filename, const char* str, int count)
{
  FILE* file = fopen(filename, "w");
```

```
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return;
  }

  fwrite(str, 1, count, file);

  if (ferror(file))
    IupMessagef("Error", "Fail when writing to file: %s", filename);

  fclose(file);
}


/********************************* Callbacks *************************************


int multitext_caret_cb (Ihandle *ih, int lin, int col)
{
  Ihandle *lbl_statusbar = IupGetDialogChild(ih, "STATUSBAR");
  IupSetfAttribute(lbl_statusbar, "TITLE", "Lin %d, Col %d", lin, col);
  return IUP_DEFAULT;
}

int item_open_action_cb(Ihandle* item_open)
{
  Ihandle* multitext = IupGetDialogChild(item_open, "MULTITEXT");
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "OPEN");
  IupSetAttribute(filedlg, "EXTFILTER", "Text Files|*.txt|All Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_open));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    char* str = read_file(filename);
    if (str)
    {
      IupSetStrAttribute(multitext, "VALUE", str);
```

```
      free(str);
    }
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_saveas_action_cb(Ihandle* item_saveas)
{
  Ihandle* multitext = IupGetDialogChild(item_saveas, "MULTITEXT");
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "SAVE");
  IupSetAttribute(filedlg, "EXTFILTER", "Text␣Files|*.txt|All␣Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_saveas));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    char* str = IupGetAttribute(multitext, "VALUE");
    int count = IupGetInt(multitext, "COUNT");
    write_file(filename, str, count);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_exit_action_cb(void)
{
  return IUP_CLOSE;
}

int goto_ok_action_cb(Ihandle* bt_ok)
{
  int line_count = IupGetInt(bt_ok, "TEXT_LINECOUNT");
  Ihandle* txt = IupGetDialogChild(bt_ok, "LINE_TEXT");
  int line = IupGetInt(txt, "VALUE");
  if (line < 1 || line >= line_count)
  {
```

```
    IupMessage("Error", "Invalid line number.");
    return IUP_DEFAULT;
  }

  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "1");
  return IUP_CLOSE;
}

int goto_cancel_action_cb(Ihandle* bt_ok)
{
  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "0");
  return IUP_CLOSE;
}

int item_goto_action_cb(Ihandle* item_goto)
{
  Ihandle* multitext = IupGetDialogChild(item_goto, "MULTITEXT");
  Ihandle *dlg, *box, *bt_ok, *bt_cancel, *txt, *lbl;

  int line_count = IupGetInt(multitext, "LINECOUNT");

  lbl = IupLabel(NULL);
  IupSetfAttribute(lbl, "TITLE", "Line Number [1-%d]:", line_count);
  txt = IupText(NULL);
  IupSetAttribute(txt, "MASK", IUP_MASK_UINT);  /* unsigned integer numbers only */
  IupSetAttribute(txt, "NAME", "LINE_TEXT");
  IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
  bt_ok = IupButton("OK", NULL);
  IupSetInt(bt_ok, "TEXT_LINECOUNT", line_count);
  IupSetAttribute(bt_ok, "PADDING", "10x2");
  IupSetCallback(bt_ok, "ACTION", (Icallback)goto_ok_action_cb);
  bt_cancel = IupButton("Cancel", NULL);
  IupSetCallback(bt_cancel, "ACTION", (Icallback)goto_cancel_action_cb);
  IupSetAttribute(bt_cancel, "PADDING", "10x2");

  box = IupVbox(
    lbl,
    txt,
    IupSetAttributes(IupHbox(
      IupFill(),
      bt_ok,
      bt_cancel,
```

```
      NULL), "NORMALIZESIZE=HORIZONTAL"),
    NULL);
  IupSetAttribute(box, "MARGIN", "10x10");
  IupSetAttribute(box, "GAP", "5");

  dlg = IupDialog(box);
  IupSetAttribute(dlg, "TITLE", "Go To Line");
  IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
  IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_ok);
  IupSetAttributeHandle(dlg, "DEFAULTESC", bt_cancel);
  IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_goto));

  IupPopup(dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(dlg, "STATUS") == 1)
  {
    int line = IupGetInt(txt, "VALUE");
    int pos;
    IupTextConvertLinColToPos(multitext, line, 0, &pos);
    IupSetInt(multitext, "CARETPOS", pos);
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }

  IupDestroy(dlg);

  return IUP_DEFAULT;
}

int find_next_action_cb(Ihandle* bt_next)
{
  Ihandle* multitext = (Ihandle*)IupGetAttribute(bt_next, "MULTITEXT");
  char* str = IupGetAttribute(multitext, "VALUE");
  int find_pos = IupGetInt(multitext, "FIND_POS");

  Ihandle* txt = IupGetDialogChild(bt_next, "FIND_TEXT");
  char* str_to_find = IupGetAttribute(txt, "VALUE");

  Ihandle* find_case = IupGetDialogChild(bt_next, "FIND_CASE");
  int casesensitive = IupGetInt(find_case, "VALUE");

  int pos = str_find(str + find_pos, str_to_find, casesensitive);
  if (pos >= 0)
```

```
    pos += find_pos;
  else if (find_pos > 0)
    pos = str_find(str, str_to_find, casesensitive);  /* try again from the start *

  if (pos >= 0)
  {
    int lin, col,
      end_pos = pos + (int)strlen(str_to_find);

    IupSetInt(multitext, "FIND_POS", end_pos);

    IupSetFocus(multitext);
    IupSetfAttribute(multitext, "SELECTIONPOS", "%d:%d", pos, end_pos);

    IupTextConvertPosToLinCol(multitext, pos, &lin, &col);
    IupTextConvertLinColToPos(multitext, lin, 0, &pos);  /* position at col=0, just
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }
  else
    IupMessage("Warning", "Text not found.");

  return IUP_DEFAULT;
}

int find_close_action_cb(Ihandle* bt_close)
{
  IupHide(IupGetDialog(bt_close));
  return IUP_DEFAULT;
}

int item_find_action_cb(Ihandle* item_find)
{
  Ihandle* dlg = (Ihandle*)IupGetAttribute(item_find, "FIND_DIALOG");
  if (!dlg)
  {
    Ihandle* multitext = IupGetDialogChild(item_find, "MULTITEXT");
    Ihandle *box, *bt_next, *bt_close, *txt, *find_case;

    txt = IupText(NULL);
    IupSetAttribute(txt, "NAME", "FIND_TEXT");
    IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
    find_case = IupToggle("Case Sensitive", NULL);
```

```
    IupSetAttribute(find_case, "NAME", "FIND_CASE");
    bt_next = IupButton("Find Next", NULL);
    IupSetAttribute(bt_next, "PADDING", "10x2");
    IupSetCallback(bt_next, "ACTION", (Icallback)find_next_action_cb);
    bt_close = IupButton("Close", NULL);
    IupSetCallback(bt_close, "ACTION", (Icallback)find_close_action_cb);
    IupSetAttribute(bt_close, "PADDING", "10x2");

    box = IupVbox(
      IupLabel("Find What:"),
      txt,
      find_case,
      IupSetAttributes(IupHbox(
        IupFill(),
        bt_next,
        bt_close,
        NULL), "NORMALIZESIZE=HORIZONTAL"),
      NULL);
    IupSetAttribute(box, "MARGIN", "10x10");
    IupSetAttribute(box, "GAP", "5");

    dlg = IupDialog(box);
    IupSetAttribute(dlg, "TITLE", "Find");
    IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
    IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_next);
    IupSetAttributeHandle(dlg, "DEFAULTESC", bt_close);
    IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_find));

    /* Save the multiline to acess it from the callbacks */
    IupSetAttribute(dlg, "MULTITEXT", (char*)multitext);

    /* Save the dialog to reuse it */
    IupSetAttribute(item_find, "FIND_DIALOG", (char*)dlg);
  }

  /* centerparent first time, next time reuse the last position */
  IupShowXY(dlg, IUP_CURRENT, IUP_CURRENT);

  return IUP_DEFAULT;
}

int item_font_action_cb(Ihandle* item_font)
```

```
{
  Ihandle* multitext = IupGetDialogChild(item_font, "MULTITEXT");
  Ihandle* fontdlg = IupFontDlg();
  char* font = IupGetAttribute(multitext, "FONT");
  IupSetStrAttribute(fontdlg, "VALUE", font);
  IupSetAttributeHandle(fontdlg, "PARENTDIALOG", IupGetDialog(item_font));

  IupPopup(fontdlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(fontdlg, "STATUS") == 1)
  {
    font = IupGetAttribute(fontdlg, "VALUE");
    IupSetStrAttribute(multitext, "FONT", font);
  }

  IupDestroy(fontdlg);
  return IUP_DEFAULT;
}

int item_about_action_cb(void)
{
  IupMessage("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣S
  return IUP_DEFAULT;
}


/********************************* Main ****************************************/


int main(int argc, char **argv)
{
  Ihandle *dlg, *vbox, *multitext, *menu;
  Ihandle *sub_menu_file, *file_menu, *item_exit, *item_open, *item_saveas, *btn_op
  Ihandle *sub_menu_edit, *edit_menu, *item_find, *item_goto, *btn_find;
  Ihandle *sub_menu_format, *format_menu, *item_font;
  Ihandle *sub_menu_help, *help_menu, *item_about;
  Ihandle *lbl_statusbar, *toolbar_hb;

  IupOpen(&argc, &argv);
  IupImageLibOpen();

  multitext = IupText(NULL);
```

```
IupSetAttribute(multitext, "MULTILINE", "YES");
IupSetAttribute(multitext, "EXPAND", "YES");
IupSetAttribute(multitext, "NAME", "MULTITEXT");

lbl_statusbar = IupLabel("Lin 1, Col 1");
IupSetAttribute(lbl_statusbar, "NAME", "STATUSBAR");
IupSetAttribute(lbl_statusbar, "EXPAND", "HORIZONTAL");
IupSetAttribute(lbl_statusbar, "PADDING", "10x5");

item_open = IupItem("Open...", NULL);
btn_open = IupButton(NULL, NULL);
IupSetAttribute(btn_open, "IMAGE", "IUP_FileOpen");
IupSetAttribute(btn_open, "FLAT", "Yes");
IupSetAttribute(btn_open, "CANFOCUS", "No");

item_saveas = IupItem("Save As...", NULL);
btn_save = IupButton(NULL, NULL);
IupSetAttribute(btn_save, "IMAGE", "IUP_FileSave");
IupSetAttribute(btn_save, "FLAT", "Yes");
IupSetAttribute(btn_save, "CANFOCUS", "No");

item_exit = IupItem("Exit", NULL);

item_find = IupItem("Find...", NULL);
btn_find = IupButton(NULL, NULL);
IupSetAttribute(btn_find, "IMAGE", "IUP_EditFind");
IupSetAttribute(btn_find, "FLAT", "Yes");
IupSetAttribute(btn_find, "CANFOCUS", "No");

toolbar_hb = IupHbox(
  btn_open,
  btn_save,
  IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
  btn_find,
  NULL);
IupSetAttribute(toolbar_hb, "MARGIN", "5x5");
IupSetAttribute(toolbar_hb, "GAP", "2");

item_goto = IupItem("Go To...", NULL);
item_font = IupItem("Font...", NULL);
item_about = IupItem("About...", NULL);
```

```
IupSetCallback(item_open, "ACTION", (Icallback)item_open_action_cb);
IupSetCallback(btn_open, "ACTION", (Icallback)item_open_action_cb);
IupSetCallback(item_saveas, "ACTION", (Icallback)item_saveas_action_cb);
IupSetCallback(btn_save, "ACTION", (Icallback)item_saveas_action_cb);
IupSetCallback(item_exit, "ACTION", (Icallback)item_exit_action_cb);
IupSetCallback(item_find, "ACTION", (Icallback)item_find_action_cb);
IupSetCallback(btn_find, "ACTION", (Icallback)item_find_action_cb);
IupSetCallback(item_goto, "ACTION", (Icallback)item_goto_action_cb);
IupSetCallback(item_font, "ACTION", (Icallback)item_font_action_cb);
IupSetCallback(item_about, "ACTION", (Icallback)item_about_action_cb);
IupSetCallback(multitext, "CARET_CB", (Icallback)multitext_caret_cb);

file_menu = IupMenu(
  item_open,
  item_saveas,
  IupSeparator(),
  item_exit,
  NULL);
edit_menu = IupMenu(
  item_find,
  item_goto,
  NULL);
format_menu = IupMenu(
  item_font,
  NULL);
help_menu = IupMenu(
  item_about,
  NULL);

sub_menu_file = IupSubmenu("File", file_menu);
sub_menu_edit = IupSubmenu("Edit", edit_menu);
sub_menu_format = IupSubmenu("Format", format_menu);
sub_menu_help = IupSubmenu("Help", help_menu);

menu = IupMenu(
  sub_menu_file,
  sub_menu_edit,
  sub_menu_format,
  sub_menu_help,
  NULL);

vbox = IupVbox(
```

```
    toolbar_hb,
    multitext,
    lbl_statusbar,
    NULL);

  dlg = IupDialog(vbox);
  IupSetAttributeHandle(dlg, "MENU", menu);
  IupSetAttribute(dlg, "TITLE", "Simple Notepad");
  IupSetAttribute(dlg, "SIZE", "HALFxHALF");

  /* parent for pre-defined dialogs in closed functions (IupMessage) */
  IupSetAttributeHandle(NULL, "PARENTDIALOG", dlg);

  IupShowXY(dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
  IupSetAttribute(dlg, "USERSIZE", NULL);

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")
require("iupluaimglib")


--********************************* Utilities *********************************


function str_find(str, str_to_find, casesensitive, start)
  if (not casesensitive) then
    return str_find(string.lower(str), string.lower(str_to_find), true, start)
  end

  return string.find(str, str_to_find, start, true)
end

function read_file(filename)
  local ifile = io.open(filename, "r")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
```

```
    return nil
  end

  local str = ifile:read("*a")
  if (not str) then
    iup.Message("Error", "Fail when reading from file: " .. filename)
    return nil
  end

  ifile:close()
  return str
end

function write_file(filename, str)
  local ifile = io.open(filename, "w")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return
  end

  if (not ifile:write(str)) then
    iup.Message("Error", "Fail when writing to file: " .. filename)
  end

  ifile:close()
end


--********************************* Main (Part 1/2) ****************************


lbl_statusbar = iup.label{title = "Lin 1, Col 1", expand = "HORIZONTAL", padding =

multitext = iup.text{
  multiline = "YES",
  expand = "YES"
}

item_open = iup.item{title="Open..."}
item_saveas = iup.item{title="Save As..."}
item_font = iup.item{title="Font..."}
item_about = iup.item{title="About..."}
```

```
item_find = iup.item{title="Find..."}
item_goto = iup.item{title="Go To..."}
item_exit = iup.item{title="Exit"}


--********************************* Callbacks *********************************


function multitext:caret_cb(lin, col)
  lbl_statusbar.title = "Lin "..lin.." , Col "..col
end

function item_open:action()
  local filedlg = iup.filedlg{
    dialogtype = "OPEN",
    filter = "*.txt",
    filterinfo = "Text Files",
    parentdialog=iup.GetDialog(self)
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    local str = read_file(filename)
    if (str) then
      multitext.value = str
    end
  end
  filedlg:destroy()
end

function item_saveas:action()
  local filedlg = iup.filedlg{
    dialogtype = "SAVE",
    filter = "*.txt",
    filterinfo = "Text Files",
    parentdialog=iup.GetDialog(self)
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)
```

```
  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    write_file(filename, multitext.value)
  end
  filedlg:destroy()
end

function item_exit:action()
  return iup.CLOSE
end

function item_goto:action()
  local line_count = multitext.linecount
  local lbl_goto = iup.label{title = "Line Number [1-"..line_count.."]:"}
  local txt_goto = iup.text{mask = iup.MASK_UINT, visiblecolumns = 20} --unsigned i

  local bt_goto_ok = iup.button{title = "OK", text_linecount = 0, padding = "10x2"}
  bt_goto_ok.text_linecount = line_count
  function bt_goto_ok:action()
    local line_count = tonumber(self.text_linecount)
    local line = tonumber(txt_goto.value)
    if (line < 1 or line >= line_count) then
      iup.Message("Error", "Invalid line number.")
      return
    end
    goto_dlg.status = 1
    return iup.CLOSE
  end

  local bt_goto_cancel = iup.button{title = "Cancel", padding = "10x2"}
  function bt_goto_cancel:action()
    goto_dlg.status = 0
    return iup.CLOSE
  end

  local box = iup.vbox{
    lbl_goto,
    txt_goto,
    iup.hbox{
      iup.fill{},
      bt_goto_ok,
      bt_goto_cancel,
```

```
      normalizesize="HORIZONTAL",
    },
    margin = "10x10",
    gap = "5",
  }
  goto_dlg = iup.dialog{
    box,
    title = "Go To Line",
    dialogframe = "Yes",
    defaultenter = bt_goto_ok,
    defaultesc = bt_goto_cancel,
    parentdialog = iup.GetDialog(self)
  }

  goto_dlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(goto_dlg.status) == 1) then
    local line = txt_goto.value
    local pos = iup.TextConvertLinColToPos(multitext, line, 0)
    multitext.caretpos = pos
    multitext.scrolltopos = pos
  end

  goto_dlg:destroy()
end

function item_find:action()
  local find_dlg = self.find_dialog
  if (not find_dlg) then
    local find_txt = iup.text{visiblecolumns = "20"}
    local find_case = iup.toggle{title = "Case Sensitive"}
    local bt_find_next = iup.button{title = "Find Next", padding = "10x2"}
    local bt_find_close = iup.button{title = "Close", padding = "10x2"}

    function bt_find_next:action()
      local find_pos = tonumber(find_dlg.find_pos)
      local str_to_find = find_txt.value

      local casesensitive = (find_case.value == "ON")

      -- test again, because it can be called from the hot key
      if (not str_to_find or str_to_find:len()==0) then
```

```
    return
  end

  if (not find_pos) then
    find_pos = 1
  end

  local str = multitext.value

  local pos, end_pos = str_find(str, str_to_find, casesensitive, find_pos)
  if (not pos) then
    pos, end_pos = str_find(str, str_to_find, casesensitive, 1) -- try again fr
  end

  if (pos) and (pos > 0) then
    pos = pos - 1
    find_dlg.find_pos = end_pos

    iup.SetFocus(multitext)
    multitext.selectionpos = pos..":"..end_pos

    local lin, col = iup.TextConvertPosToLinCol(multitext, pos)
    local pos = iup.TextConvertLinColToPos(multitext, lin, 0) -- position at c
    multitext.scrolltopos = pos
  else
    find_dlg.find_pos = nil
    iup.Message("Warning", "Text not found.")
  end
end

function bt_find_close:action()
  iup.Hide(iup.GetDialog(self)) -- do not destroy, just hide
end

box = iup.vbox{
  iup.label{title = "Find What:"},
  find_txt,
  find_case,
  iup.hbox{
    iup.fill{},
    bt_find_next,
    bt_find_close,
```

```
        normalizesize="HORIZONTAL",
      },
      margin = "10x10",
      gap = "5",
    }

    find_dlg = iup.dialog{
      box,
      title = "Find",
      dialogframe = "Yes",
      defaultenter = bt_next,
      defaultesc = bt_close,
      parentdialog = iup.GetDialog(self)
      }

    -- Save the dialog to reuse it
    self.find_dialog = find_dlg -- from the main dialog */
  end

  -- centerparent first time, next time reuse the last position
  find_dlg:showxy(iup.CURRENT, iup.CURRENT)
end

function item_font:action()
  local font = multitext.font
  local fontdlg = iup.fontdlg{value = font, parentdialog=iup.GetDialog(self)}

  fontdlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(fontdlg.status) == 1) then
    multitext.font = fontdlg.value
  end

  fontdlg:destroy()
end

function item_about:action()
  iup.Message("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣
end


--********************************* Main (Part 2/2) ****************************
```

```
file_menu = iup.menu{item_open,item_saveas,iup.separator{},item_exit}
edit_menu = iup.menu{item_find, item_goto}
format_menu = iup.menu{item_font}
help_menu = iup.menu{item_about}
sub_menu_file = iup.submenu{file_menu, title = "File"}
sub_menu_edit = iup.submenu{edit_menu, title = "Edit"}
sub_menu_format = iup.submenu{format_menu, title = "Format"}
sub_menu_help = iup.submenu{help_menu, title = "Help"}

menu = iup.menu{
  sub_menu_file,
  sub_menu_edit,
  sub_menu_format,
  sub_menu_help,
  }

btn_open = iup.button{image = "IUP_FileOpen", flat = "Yes", action = item_open.actio
btn_save = iup.button{image = "IUP_FileSave", flat = "Yes", action = item_saveas.ac
btn_find = iup.button{image = "IUP_EditFind", flat = "Yes", action = item_find.actio

toolbar_hb = iup.hbox{
  btn_open,
  btn_save,
  iup.label{separator="VERTICAL"},
  btn_find,
  margin = "5x5",
  gap = 2,
  }

vbox = iup.vbox{
  toolbar_hb,
  multitext,
  lbl_statusbar,
}

dlg = iup.dialog{
  vbox,
  title = "Simple Notepad",
  size = "HALFxHALF",
  menu = menu
```

```
}

-- parent for pre-defined dialogs in closed functions (IupMessage)
iup.SetGlobal("PARENTDIALOG", iup.SetHandleName(dlg))

dlg:showxy(iup.CENTER, iup.CENTER)
dlg.usersize = nil

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

首先是 *multitext_caret_cb* 回调函数的变化。我们使用了这个回调传回的参数来设置状态栏。首先我们使用 *IupGetDialogChild* 函数来获取状态栏上的 *IupLabel* 控件句柄，接着我们用 *lin* 和 *col* 参数来构造一个字符串，并将这个字符串设置为标签的 *TITLE* 属性值。

接着，看其它发生变化的地方。在 *IupOpen* 函数之后，我们调用了 *IupImageLibOpen* 函数，这个函数载入了我们在工具栏上的按钮使用的图像。

接着，间隔没有多少行，我们可以找到状态栏的定义。我们为状态栏定义了一个标签，并用它来显示提示信息。我们把他的 *EXPAND* 属性设置为 *HORIZONTAL*，这样它就会占用垂直盒子的所有水平空间。接着，我们可以看到一些按钮定义和一些属性设置。我们使用 *IupSetAttribute* 为按钮设置图像。图像的名称可以在 *IupImageLib* 文档中找到。接着，出现的是一个 *IupHbox*，它包含了这个按钮。间隔一些行，我们为这些按钮设置了和对应菜单项目一样的回调函数。这是显而易见的，毕竟两者是同一行为的不同访问形式。我们还为按钮设置 *FLAT* 属性，这样它们的边框就被移除了，按钮看起来也更像一个工具栏的按钮。我们设置按钮的 *CANFOCUS* 为 *No*，使它们不能获得输入焦点。

最后的变化在 *vbox* 中，垂直盒子里已经包含了我们的文本框。现在我们将工具栏放在文本框之前，把状态栏放在文本框之后，然后我们就具有了一个带有工具栏和状态栏的文本编辑器。在下一节，我们会为我们的菜单添加热键。

3.6 定义热键

程序可以为用户提供热键来快速访问菜单项。我们可以通过 *IupDialog* 的 *K_ANY* 回调来定义热键。这个回调在键盘事件发生时被调用。**IUP** 也可以为指定的组合键提供回调函数。举个例子，我们要为文件打开菜单项目指定组合键 *Ctrl+O*，只需要设置 *K_cO* 回调即可。*Keyboard Codes* 显示了完整的 **IUP** 按键映射列表。

C语言

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <iup.h>
```

```
/********************************* Utilities ***********************************/

int str_compare(const char *l, const char *r, int casesensitive)
{
  if (!l || !r)
    return 0;

  while (*l && *r)
  {
    int diff;
    char l_char = *l,
      r_char = *r;

    /* compute the difference of both characters */
    if (casesensitive)
      diff = l_char - r_char;
    else
      diff = tolower((int)l_char) - tolower((int)r_char);

    /* if they differ we have a result */
    if (diff != 0)
      return 0;

    /* otherwise process the next characters */
    ++l;
    ++r;
  }

  /* check also for terminator */
  if (*l == *r)
    return 1;

  if (*r == 0)
    return 1;  /* if second string is at terminator, then it is partially equal */

  return 0;
}

int str_find(const char *str, const char *str_to_find, int casesensitive)
```

```
{
  int i, str_len, str_to_find_len, count;

  if (!str || str[0] == 0 || !str_to_find || str_to_find[0] == 0)
    return -1;

  str_len = (int)strlen(str);
  str_to_find_len = (int)strlen(str_to_find);
  count = str_len - str_to_find_len;
  if (count < 0)
    return -1;

  count++;

  for (i = 0; i<count; i++)
  {
    if (str_compare(str, str_to_find, casesensitive))
      return i;

    str++;
  }

  return -1;
}

char* read_file(const char* filename)
{
  int size;
  char* str;
  FILE* file = fopen(filename, "rb");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return NULL;
  }

  /* calculate file size */
  fseek(file, 0, SEEK_END);
  size = ftell(file);
  fseek(file, 0, SEEK_SET);

  /* allocate memory for the file contents + nul terminator */
```

```
  str = malloc(size + 1);
  /* read all data at once */
  fread(str, size, 1, file);
  /* set the nul terminator */
  str[size] = 0;

  if (ferror(file))
    IupMessagef("Error", "Fail when reading from file: %s", filename);

  fclose(file);
  return str;
}

void write_file(const char* filename, const char* str, int count)
{
  FILE* file = fopen(filename, "w");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return;
  }

  fwrite(str, 1, count, file);

  if (ferror(file))
    IupMessagef("Error", "Fail when writing to file: %s", filename);

  fclose(file);
}


/********************************* Callbacks *************************************


int multitext_caret_cb(Ihandle *ih, int lin, int col)
{
  Ihandle *lbl_statusbar = IupGetDialogChild(ih, "STATUSBAR");
  IupSetfAttribute(lbl_statusbar, "TITLE", "Lin %d, Col %d", lin, col);
  return IUP_DEFAULT;
}

int item_open_action_cb(Ihandle* item_open)
```

```
{
  Ihandle* multitext = IupGetDialogChild(item_open, "MULTITEXT");
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "OPEN");
  IupSetAttribute(filedlg, "EXTFILTER", "Text␣Files|*.txt|All␣Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_open));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    char* str = read_file(filename);
    if (str)
    {
      IupSetStrAttribute(multitext, "VALUE", str);
      free(str);
    }
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_saveas_action_cb(Ihandle* item_saveas)
{
  Ihandle* multitext = IupGetDialogChild(item_saveas, "MULTITEXT");
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "SAVE");
  IupSetAttribute(filedlg, "EXTFILTER", "Text␣Files|*.txt|All␣Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_saveas));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    char* str = IupGetAttribute(multitext, "VALUE");
    int count = IupGetInt(multitext, "COUNT");
    write_file(filename, str, count);
  }
```

```
  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_exit_action_cb(void)
{
  return IUP_CLOSE;
}

int goto_ok_action_cb(Ihandle* bt_ok)
{
  int line_count = IupGetInt(bt_ok, "TEXT_LINECOUNT");
  Ihandle* txt = IupGetDialogChild(bt_ok, "LINE_TEXT");
  int line = IupGetInt(txt, "VALUE");
  if (line < 1 || line >= line_count)
  {
    IupMessage("Error", "Invalid line number.");
    return IUP_DEFAULT;
  }

  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "1");
  return IUP_CLOSE;
}

int goto_cancel_action_cb(Ihandle* bt_ok)
{
  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "0");
  return IUP_CLOSE;
}

int item_goto_action_cb(Ihandle* item_goto)
{
  Ihandle* multitext = IupGetDialogChild(item_goto, "MULTITEXT");
  Ihandle *dlg, *box, *bt_ok, *bt_cancel, *txt, *lbl;

  int line_count = IupGetInt(multitext, "LINECOUNT");

  lbl = IupLabel(NULL);
  IupSetfAttribute(lbl, "TITLE", "Line Number [1-%d]:", line_count);
  txt = IupText(NULL);
  IupSetAttribute(txt, "MASK", IUP_MASK_UINT);  /* unsigned integer numbers only */
  IupSetAttribute(txt, "NAME", "LINE_TEXT");
```

```
  IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
  bt_ok = IupButton("OK", NULL);
  IupSetInt(bt_ok, "TEXT_LINECOUNT", line_count);
  IupSetAttribute(bt_ok, "PADDING", "10x2");
  IupSetCallback(bt_ok, "ACTION", (Icallback)goto_ok_action_cb);
  bt_cancel = IupButton("Cancel", NULL);
  IupSetCallback(bt_cancel, "ACTION", (Icallback)goto_cancel_action_cb);
  IupSetAttribute(bt_cancel, "PADDING", "10x2");

  box = IupVbox(
    lbl,
    txt,
    IupSetAttributes(IupHbox(
      IupFill(),
      bt_ok,
      bt_cancel,
      NULL), "NORMALIZESIZE=HORIZONTAL"),
    NULL);
  IupSetAttribute(box, "MARGIN", "10x10");
  IupSetAttribute(box, "GAP", "5");

  dlg = IupDialog(box);
  IupSetAttribute(dlg, "TITLE", "Go␣To␣Line");
  IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
  IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_ok);
  IupSetAttributeHandle(dlg, "DEFAULTESC", bt_cancel);
  IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_goto));

  IupPopup(dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(dlg, "STATUS") == 1)
  {
    int line = IupGetInt(txt, "VALUE");
    int pos;
    IupTextConvertLinColToPos(multitext, line, 0, &pos);
    IupSetInt(multitext, "CARETPOS", pos);
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }

  IupDestroy(dlg);

  return IUP_DEFAULT;
```

```
}

int find_next_action_cb(Ihandle* bt_next)
{
  Ihandle* multitext = (Ihandle*)IupGetAttribute(bt_next, "MULTITEXT");
  char* str = IupGetAttribute(multitext, "VALUE");
  int find_pos = IupGetInt(multitext, "FIND_POS");

  Ihandle* txt = IupGetDialogChild(bt_next, "FIND_TEXT");
  char* str_to_find = IupGetAttribute(txt, "VALUE");

  Ihandle* find_case = IupGetDialogChild(bt_next, "FIND_CASE");
  int casesensitive = IupGetInt(find_case, "VALUE");

  int pos = str_find(str + find_pos, str_to_find, casesensitive);
  if (pos >= 0)
    pos += find_pos;
  else if (find_pos > 0)
    pos = str_find(str, str_to_find, casesensitive);  /* try again from the start *

  if (pos >= 0)
  {
    int lin, col,
      end_pos = pos + (int)strlen(str_to_find);

    IupSetInt(multitext, "FIND_POS", end_pos);

    IupSetFocus(multitext);
    IupSetfAttribute(multitext, "SELECTIONPOS", "%d:%d", pos, end_pos);

    IupTextConvertPosToLinCol(multitext, pos, &lin, &col);
    IupTextConvertLinColToPos(multitext, lin, 0, &pos);  /* position at col=0, just
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }
  else
    IupMessage("Warning", "Text not found.");

  return IUP_DEFAULT;
}

int find_close_action_cb(Ihandle* bt_close)
{
```

```
  IupHide(IupGetDialog(bt_close));
  return IUP_DEFAULT;
}

int item_find_action_cb(Ihandle* item_find)
{
  Ihandle* dlg = (Ihandle*)IupGetAttribute(item_find, "FIND_DIALOG");
  if (!dlg)
  {
    Ihandle* multitext = IupGetDialogChild(item_find, "MULTITEXT");
    Ihandle *box, *bt_next, *bt_close, *txt, *find_case;

    txt = IupText(NULL);
    IupSetAttribute(txt, "NAME", "FIND_TEXT");
    IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
    find_case = IupToggle("Case Sensitive", NULL);
    IupSetAttribute(find_case, "NAME", "FIND_CASE");
    bt_next = IupButton("Find Next", NULL);
    IupSetAttribute(bt_next, "PADDING", "10x2");
    IupSetCallback(bt_next, "ACTION", (Icallback)find_next_action_cb);
    bt_close = IupButton("Close", NULL);
    IupSetCallback(bt_close, "ACTION", (Icallback)find_close_action_cb);
    IupSetAttribute(bt_close, "PADDING", "10x2");

    box = IupVbox(
      IupLabel("Find What:"),
      txt,
      find_case,
      IupSetAttributes(IupHbox(
        IupFill(),
        bt_next,
        bt_close,
        NULL), "NORMALIZESIZE=HORIZONTAL"),
      NULL);
    IupSetAttribute(box, "MARGIN", "10x10");
    IupSetAttribute(box, "GAP", "5");

    dlg = IupDialog(box);
    IupSetAttribute(dlg, "TITLE", "Find");
    IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
    IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_next);
    IupSetAttributeHandle(dlg, "DEFAULTESC", bt_close);
```

```
    IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_find));

    /* Save the multiline to acess it from the callbacks */
    IupSetAttribute(dlg, "MULTITEXT", (char*)multitext);

    /* Save the dialog to reuse it */
    IupSetAttribute(item_find, "FIND_DIALOG", (char*)dlg);
  }

  /* centerparent first time, next time reuse the last position */
  IupShowXY(dlg, IUP_CURRENT, IUP_CURRENT);

  return IUP_DEFAULT;
}

int item_font_action_cb(Ihandle* item_font)
{
  Ihandle* multitext = IupGetDialogChild(item_font, "MULTITEXT");
  Ihandle* fontdlg = IupFontDlg();
  char* font = IupGetAttribute(multitext, "FONT");
  IupSetStrAttribute(fontdlg, "VALUE", font);
  IupSetAttributeHandle(fontdlg, "PARENTDIALOG", IupGetDialog(item_font));

  IupPopup(fontdlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(fontdlg, "STATUS") == 1)
  {
    font = IupGetAttribute(fontdlg, "VALUE");
    IupSetStrAttribute(multitext, "FONT", font);
  }

  IupDestroy(fontdlg);
  return IUP_DEFAULT;
}

int item_about_action_cb(void)
{
  IupMessage("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣S
  return IUP_DEFAULT;
}
```

```
/********************************** Main ****************************************/


int main(int argc, char **argv)
{
  Ihandle *dlg, *vbox, *multitext, *menu;
  Ihandle *sub_menu_file, *file_menu, *item_exit, *item_open, *item_saveas, *btn_op
  Ihandle *sub_menu_edit, *edit_menu, *item_find, *item_goto, *btn_find;
  Ihandle *sub_menu_format, *format_menu, *item_font;
  Ihandle *sub_menu_help, *help_menu, *item_about;
  Ihandle *lbl_statusbar, *toolbar_hb;

  IupOpen(&argc, &argv);
  IupImageLibOpen();

  multitext = IupText(NULL);
  IupSetAttribute(multitext, "MULTILINE", "YES");
  IupSetAttribute(multitext, "EXPAND", "YES");
  IupSetAttribute(multitext, "NAME", "MULTITEXT");

  lbl_statusbar = IupLabel("Lin 1, Col 1");
  IupSetAttribute(lbl_statusbar, "NAME", "STATUSBAR");
  IupSetAttribute(lbl_statusbar, "EXPAND", "HORIZONTAL");
  IupSetAttribute(lbl_statusbar, "PADDING", "10x5");

  item_open = IupItem("&Open...\tCtrl+O", NULL);
  btn_open = IupButton(NULL, NULL);
  IupSetAttribute(btn_open, "IMAGE", "IUP_FileOpen");
  IupSetAttribute(btn_open, "FLAT", "Yes");
  IupSetAttribute(btn_open, "TIP", "Open (Ctrl+O)");
  IupSetAttribute(btn_open, "CANFOCUS", "No");

  item_saveas = IupItem("Save &As...\tCtrl+S", NULL);
  btn_save = IupButton(NULL, NULL);
  IupSetAttribute(btn_save, "IMAGE", "IUP_FileSave");
  IupSetAttribute(btn_save, "FLAT", "Yes");
  IupSetAttribute(btn_save, "TIP", "Save (Ctrl+S)");
  IupSetAttribute(btn_save, "CANFOCUS", "No");

  item_exit = IupItem("E&xit", NULL);

  item_find = IupItem("&Find...\tCtrl+F", NULL);
```

```
  btn_find = IupButton(NULL, NULL);
  IupSetAttribute(btn_find, "IMAGE", "IUP_EditFind");
  IupSetAttribute(btn_find, "FLAT", "Yes");
  IupSetAttribute(btn_find, "TIP", "Find (Ctrl+F)");
  IupSetAttribute(btn_find, "CANFOCUS", "No");

  toolbar_hb = IupHbox(
    btn_open,
    btn_save,
    IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
    btn_find,
    NULL);
  IupSetAttribute(toolbar_hb, "MARGIN", "5x5");
  IupSetAttribute(toolbar_hb, "GAP", "2");

  item_goto = IupItem("&Go To...\tCtrl+G", NULL);
  item_font = IupItem("&Font...", NULL);
  item_about = IupItem("&About...", NULL);

  IupSetCallback(item_open, "ACTION", (Icallback)item_open_action_cb);
  IupSetCallback(btn_open, "ACTION", (Icallback)item_open_action_cb);
  IupSetCallback(item_saveas, "ACTION", (Icallback)item_saveas_action_cb);
  IupSetCallback(btn_save, "ACTION", (Icallback)item_saveas_action_cb);
  IupSetCallback(item_exit, "ACTION", (Icallback)item_exit_action_cb);
  IupSetCallback(item_find, "ACTION", (Icallback)item_find_action_cb);
  IupSetCallback(btn_find, "ACTION", (Icallback)item_find_action_cb);
  IupSetCallback(item_goto, "ACTION", (Icallback)item_goto_action_cb);
  IupSetCallback(item_font, "ACTION", (Icallback)item_font_action_cb);
  IupSetCallback(item_about, "ACTION", (Icallback)item_about_action_cb);
  IupSetCallback(multitext, "CARET_CB", (Icallback)multitext_caret_cb);

  file_menu = IupMenu(
    item_open,
    item_saveas,
    IupSeparator(),
    item_exit,
    NULL);
  edit_menu = IupMenu(
    item_find,
    item_goto,
    NULL);
  format_menu = IupMenu(
```

```
    item_font,
    NULL);
  help_menu = IupMenu(
    item_about,
    NULL);

  sub_menu_file = IupSubmenu("&File", file_menu);
  sub_menu_edit = IupSubmenu("&Edit", edit_menu);
  sub_menu_format = IupSubmenu("F&ormat", format_menu);
  sub_menu_help = IupSubmenu("&Help", help_menu);

  menu = IupMenu(
    sub_menu_file,
    sub_menu_edit,
    sub_menu_format,
    sub_menu_help,
    NULL);

  vbox = IupVbox(
    toolbar_hb,
    multitext,
    lbl_statusbar,
    NULL);

  dlg = IupDialog(vbox);
  IupSetAttributeHandle(dlg, "MENU", menu);
  IupSetAttribute(dlg, "TITLE", "Simple Notepad");
  IupSetAttribute(dlg, "SIZE", "HALFxHALF");

  /* parent for pre-defined dialogs in closed functions (IupMessage) */
  IupSetAttributeHandle(NULL, "PARENTDIALOG", dlg);

  IupSetCallback(dlg, "K_cO", (Icallback)item_open_action_cb);
  IupSetCallback(dlg, "K_cS", (Icallback)item_saveas_action_cb);
  IupSetCallback(dlg, "K_cF", (Icallback)item_find_action_cb);
  IupSetCallback(dlg, "K_cG", (Icallback)item_goto_action_cb);

  IupShowXY(dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
  IupSetAttribute(dlg, "USERSIZE", NULL);

  IupMainLoop();
```

```
  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")
require("iupluaimglib")


--********************************* Utilities *********************************


function str_find(str, str_to_find, casesensitive, start)
  if (not casesensitive) then
    return str_find(string.lower(str), string.lower(str_to_find), true, start)
  end

  return string.find(str, str_to_find, start, true)
end

function read_file(filename)
  local ifile = io.open(filename, "r")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return nil
  end

  local str = ifile:read("*a")
  if (not str) then
    iup.Message("Error", "Fail when reading from file: " .. filename)
    return nil
  end

  ifile:close()
  return str
end

function write_file(filename, str)
  local ifile = io.open(filename, "w")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return false
```

```
  end

  if (not ifile:write(str)) then
    iup.Message("Error", "Fail when writing to file: " .. filename)
  end

  ifile:close()
  return true
end


--********************************* Main (Part 1/2) ****************************


lbl_statusbar = iup.label{title = "Lin 1, Col 1", expand = "HORIZONTAL", padding =

multitext = iup.text{
  multiline = "YES",
  expand = "YES"
}

item_open = iup.item{title = "&Open...\tCtrl+O"}
item_saveas = iup.item{title="Save &As...\tCtrl+S"}
item_font = iup.item{title="&Font..."}
item_about = iup.item{title="&About..."}
item_find = iup.item{title="&Find...\tCtrl+F"}
item_goto = iup.item{title="&Go To..."}
item_exit = iup.item{title="E&xit"}


--********************************* Callbacks **********************************


function multitext:caret_cb(lin, col)
  lbl_statusbar.title = "Lin "..lin..", Col "..col
end

function item_open:action()
  local filedlg = iup.filedlg{
    dialogtype = "OPEN",
    filter = "*.txt",
    filterinfo = "Text Files",
```

```
    parentdialog=iup.GetDialog(self)
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    local str = read_file(filename)
    if (str) then
      multitext.value = str
    end
  end
  filedlg:destroy()
end

function item_saveas:action()
  local filedlg = iup.filedlg{
    dialogtype = "SAVE",
    filter = "*.txt",
    filterinfo = "Text Files",
    parentdialog=iup.GetDialog(self)
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    write_file(filename, multitext.value)
  end
  filedlg:destroy()
end

function item_exit:action()
  return iup.CLOSE
end

function item_goto:action()
  local line_count = multitext.linecount
  local lbl_goto = iup.label{title = "Line Number [1-"..line_count.."]:"}
  local txt_goto = iup.text{mask = iup.MASK_UINT, visiblecolumns = 20} --unsigned i

  local bt_goto_ok = iup.button{title = "OK", text_linecount = 0, padding = "10x2"}
```

```
bt_goto_ok.text_linecount = line_count
function bt_goto_ok:action()
  local line_count = tonumber(self.text_linecount)
  local line = tonumber(txt_goto.value)
  if (line < 1 or line >= line_count) then
    iup.Message("Error", "Invalid line number.")
    return
  end
  goto_dlg.status = 1
  return iup.CLOSE
end

local bt_goto_cancel = iup.button{title = "Cancel", padding = "10x2"}
function bt_goto_cancel:action()
  goto_dlg.status = 0
  return iup.CLOSE
end

local box = iup.vbox{
  lbl_goto,
  txt_goto,
  iup.hbox{
    iup.fill{},
    bt_goto_ok,
    bt_goto_cancel,
    normalizesize="HORIZONTAL",
  },
  margin = "10x10",
  gap = "5",
}
goto_dlg = iup.dialog{
  box,
  title = "Go To Line",
  dialogframe = "Yes",
  defaultenter = bt_goto_ok,
  defaultesc = bt_goto_cancel,
  parentdialog = iup.GetDialog(self)
}

goto_dlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

if (tonumber(goto_dlg.status) == 1) then
```

```
    local line = txt_goto.value
    local pos = iup.TextConvertLinColToPos(multitext, line, 0)
    multitext.caretpos = pos
    multitext.scrolltopos = pos
  end

  goto_dlg:destroy()
end

function item_find:action()
  local find_dlg = self.find_dialog
  if (not find_dlg) then
    local find_txt = iup.text{visiblecolumns = "20"}
    local find_case = iup.toggle{title = "Case Sensitive"}
    local bt_find_next = iup.button{title = "Find Next", padding = "10x2"}
    local bt_find_close = iup.button{title = "Close", padding = "10x2"}

    function bt_find_next:action()
      local find_pos = tonumber(find_dlg.find_pos)
      local str_to_find = find_txt.value

      local casesensitive = (find_case.value == "ON")

      -- test again, because it can be called from the hot key
      if (not str_to_find or str_to_find:len()==0) then
        return
      end

      if (not find_pos) then
        find_pos = 1
      end

      local str = multitext.value

      local pos, end_pos = str_find(str, str_to_find, casesensitive, find_pos)
      if (not pos) then
        pos, end_pos = str_find(str, str_to_find, casesensitive, 1)  -- try again fr
      end

      if (pos) and (pos > 0) then
        pos = pos - 1
        find_dlg.find_pos = end_pos
```

```
    iup.SetFocus(multitext)
    multitext.selectionpos = pos..":"..end_pos

    local lin, col = iup.TextConvertPosToLinCol(multitext, pos)
    local pos = iup.TextConvertLinColToPos(multitext, lin, 0) -- position at c
    multitext.scrolltopos = pos
  else
    find_dlg.find_pos = nil
    iup.Message("Warning", "Text not found.")
  end
end

function bt_find_close:action()
  iup.Hide(iup.GetDialog(self)) -- do not destroy, just hide
end

box = iup.vbox{
  iup.label{title = "Find What:"},
  find_txt,
  find_case,
  iup.hbox{
    iup.fill{},
    bt_find_next,
    bt_find_close,
    normalizesize="HORIZONTAL",
  },
  margin = "10x10",
  gap = "5",
}

find_dlg = iup.dialog{
  box,
  title = "Find",
  dialogframe = "Yes",
  defaultenter = bt_next,
  defaultesc = bt_close,
  parentdialog = iup.GetDialog(self)
  }

-- Save the dialog to reuse it
self.find_dialog = find_dlg -- from the main dialog */
```

```
  end

  -- centerparent first time, next time reuse the last position
  find_dlg:showxy(iup.CURRENT, iup.CURRENT)
end

function item_font:action()
  local font = multitext.font
  local fontdlg = iup.fontdlg{value = font, parentdialog=iup.GetDialog(self)}

  fontdlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(fontdlg.status) == 1) then
    multitext.font = fontdlg.value
  end

  fontdlg:destroy()
end

function item_about:action()
  iup.Message("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣
end


--********************************* Main (Part 2/2) ****************************


file_menu = iup.menu{item_open,item_saveas,iup.separator{},item_exit}
edit_menu = iup.menu{item_find, item_goto}
format_menu = iup.menu{item_font}
help_menu = iup.menu{item_about}
sub_menu_file = iup.submenu{file_menu, title = "&File"}
sub_menu_edit = iup.submenu{edit_menu, title = "&Edit"}
sub_menu_format = iup.submenu{format_menu, title = "F&ormat"}
sub_menu_help = iup.submenu{help_menu, title = "&Help"}

menu = iup.menu{
  sub_menu_file,
  sub_menu_edit,
  sub_menu_format,
  sub_menu_help,
  }
```

```
btn_open = iup.button{image = "IUP_FileOpen", flat = "Yes", action = item_open.actio
btn_save = iup.button{image = "IUP_FileSave", flat = "Yes", action = item_saveas.ac
btn_find = iup.button{image = "IUP_EditFind", flat = "Yes", action = item_find.actio

toolbar_hb = iup.hbox{
  btn_open,
  btn_save,
  iup.label{separator="VERTICAL"},
  btn_find,
  margin = "5x5",
  gap = 2,
  }

vbox = iup.vbox{
  toolbar_hb,
  multitext,
  lbl_statusbar,
}

dlg = iup.dialog{
  vbox,
  title = "Simple Notepad",
  size = "HALFxHALF",
  menu = menu
}

function dlg:k_any(c)
  if (c == iup.K_cO) then
    item_open:action()
  elseif (c == iup.K_cS) then
    item_saveas:action()
  elseif (c == iup.K_cF) then
    item_find:action()
  elseif (c == iup.K_cG) then
    item_goto:action()
  end
end

-- parent for pre-defined dialogs in closed functions (IupMessage)
iup.SetGlobal("PARENTDIALOG", iup.SetHandleName(dlg))
```

```
dlg:showxy(iup.CENTER,iup.CENTER)
dlg.usersize = nil

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

本例的变化不大。我们仅仅为每个菜单项添加 *Ctrl+?* 的热键。

在之后的一些行，我们使用 *IupSetCallback* 函数指定了组合键的回调函数。

为了提高用户体验，我们还可以在菜单项目的文本中使用 *&* 后跟案件字母来为菜单项目指定快捷键。对于主菜单，我们可以使用 *Alt+key* 来访问主菜单项目，其中 *key* 为目标菜单项的 *&* 后字母。举个例子，我们可以使用 *Alt+F* 来访问文件菜单，一个菜单项展开它的子菜单后，我们直接使用 *&* 后的字母就可以访问该菜单项目。在用 *Alt+F* 打开文件菜单后，我们只需要按下 *X* 就可以退出程序。

最后，我们为工具栏按钮添加了 *TIP* 属性，这样当鼠标停留在这些按钮上就会显示一些关于这个按钮的一些辅助信息。

3.7 最近访问文件菜单和配置文件

许多文本编辑器都提供访问最近打开文件的菜单项。在这里，我们使用 *IupConfig* 来实现这个功能。*IupConfig* 可以用来存储配置信息。它实现了一组用于载入，存储程序配置变量的函数。举个例子：最近文件列表，窗口的最后位置和大小，程序关闭前的配置信息，这个都可以使用 *IupConfig* 来存储。每一个变量具有一个名字，一个值，以及它所属的组。

为了使用 *IupConfig*，我们需要包含 *iup_config.h* 头文件。

C语言

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <iup.h>
#include <iup_config.h>


/********************************** Utilities *************************************


int str_compare(const char *l, const char *r, int casesensitive)
{
  if (!l || !r)
    return 0;
```

```
  while (*l && *r)
  {
    int diff;
    char l_char = *l,
      r_char = *r;

    /* compute the difference of both characters */
    if (casesensitive)
      diff = l_char - r_char;
    else
      diff = tolower((int)l_char) - tolower((int)r_char);

    /* if they differ we have a result */
    if (diff != 0)
      return 0;

    /* otherwise process the next characters */
    ++l;
    ++r;
  }

  /* check also for terminator */
  if (*l == *r)
    return 1;

  if (*r == 0)
    return 1;  /* if second string is at terminator, then it is partially equal */

  return 0;
}

int str_find(const char *str, const char *str_to_find, int casesensitive)
{
  int i, str_len, str_to_find_len, count;

  if (!str || str[0] == 0 || !str_to_find || str_to_find[0] == 0)
    return -1;

  str_len = (int)strlen(str);
  str_to_find_len = (int)strlen(str_to_find);
  count = str_len - str_to_find_len;
  if (count < 0)
```

```
    return -1;

  count++;

  for (i = 0; i<count; i++)
  {
    if (str_compare(str, str_to_find, casesensitive))
      return i;

    str++;
  }

  return -1;
}

char* read_file(const char* filename)
{
  int size;
  char* str;
  FILE* file = fopen(filename, "rb");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return NULL;
  }

  /* calculate file size */
  fseek(file, 0, SEEK_END);
  size = ftell(file);
  fseek(file, 0, SEEK_SET);

  /* allocate memory for the file contents + nul terminator */
  str = malloc(size + 1);
  /* read all data at once */
  fread(str, size, 1, file);
  /* set the nul terminator */
  str[size] = 0;

  if (ferror(file))
    IupMessagef("Error", "Fail when reading from file: %s", filename);

  fclose(file);
```

```
  return str;
}

int write_file(const char* filename, const char* str, int count)
{
  FILE* file = fopen(filename, "w");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return 0;
  }

  fwrite(str, 1, count, file);

  if (ferror(file))
    IupMessagef("Error", "Fail when writing to file: %s", filename);

  fclose(file);
  return 1;
}


/********************************* Callbacks *************************************


int config_recent_cb(Ihandle* ih)
{
  char* filename = IupGetAttribute(ih, "TITLE");
  char* str = read_file(filename);
  if (str)
  {
    Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");
    IupSetStrAttribute(multitext, "VALUE", str);
    free(str);
  }
  return IUP_DEFAULT;
}

int multitext_caret_cb(Ihandle *ih, int lin, int col)
{
  Ihandle *lbl_statusbar = IupGetDialogChild(ih, "STATUSBAR");
  IupSetfAttribute(lbl_statusbar, "TITLE", "Lin %d, Col %d", lin, col);
```

```
  return IUP_DEFAULT;
}

int item_open_action_cb(Ihandle* item_open)
{
  Ihandle* multitext = IupGetDialogChild(item_open, "MULTITEXT");
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "OPEN");
  IupSetAttribute(filedlg, "EXTFILTER", "Text␣Files|*.txt|All␣Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_open));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    char* str = read_file(filename);
    if (str)
    {
      Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
      IupConfigRecentUpdate(config, filename);

      IupSetStrAttribute(multitext, "VALUE", str);
      free(str);
    }
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_saveas_action_cb(Ihandle* item_saveas)
{
  Ihandle* multitext = IupGetDialogChild(item_saveas, "MULTITEXT");
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "SAVE");
  IupSetAttribute(filedlg, "EXTFILTER", "Text␣Files|*.txt|All␣Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_saveas));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(filedlg, "STATUS") != -1)
```

```
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
    char* filename = IupGetAttribute(filedlg, "VALUE");
    char* str = IupGetAttribute(multitext, "VALUE");
    int count = IupGetInt(multitext, "COUNT");
    if (write_file(filename, str, count))
      IupConfigRecentUpdate(config, filename);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_exit_action_cb(Ihandle* item_exit)
{
  Ihandle* dlg = IupGetDialog(item_exit);
  Ihandle* config = (Ihandle*)IupGetAttribute(dlg, "CONFIG");
  IupConfigDialogClosed(config, dlg, "MainWindow");
  IupConfigSave(config);
  IupDestroy(config);
  return IUP_CLOSE;
}

int goto_ok_action_cb(Ihandle* bt_ok)
{
  int line_count = IupGetInt(bt_ok, "TEXT_LINECOUNT");
  Ihandle* txt = IupGetDialogChild(bt_ok, "LINE_TEXT");
  int line = IupGetInt(txt, "VALUE");
  if (line < 1 || line >= line_count)
  {
    IupMessage("Error", "Invalid line number.");
    return IUP_DEFAULT;
  }

  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "1");
  return IUP_CLOSE;
}

int goto_cancel_action_cb(Ihandle* bt_ok)
{
  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "0");
  return IUP_CLOSE;
```

```
}

int item_goto_action_cb(Ihandle* item_goto)
{
  Ihandle* multitext = IupGetDialogChild(item_goto, "MULTITEXT");
  Ihandle *dlg, *box, *bt_ok, *bt_cancel, *txt, *lbl;

  int line_count = IupGetInt(multitext, "LINECOUNT");

  lbl = IupLabel(NULL);
  IupSetfAttribute(lbl, "TITLE", "Line Number [1-%d]:", line_count);
  txt = IupText(NULL);
  IupSetAttribute(txt, "MASK", IUP_MASK_UINT);  /* unsigned integer numbers only */
  IupSetAttribute(txt, "NAME", "LINE_TEXT");
  IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
  bt_ok = IupButton("OK", NULL);
  IupSetInt(bt_ok, "TEXT_LINECOUNT", line_count);
  IupSetAttribute(bt_ok, "PADDING", "10x2");
  IupSetCallback(bt_ok, "ACTION", (Icallback)goto_ok_action_cb);
  bt_cancel = IupButton("Cancel", NULL);
  IupSetCallback(bt_cancel, "ACTION", (Icallback)goto_cancel_action_cb);
  IupSetAttribute(bt_cancel, "PADDING", "10x2");

  box = IupVbox(
    lbl,
    txt,
    IupSetAttributes(IupHbox(
      IupFill(),
      bt_ok,
      bt_cancel,
      NULL), "NORMALIZESIZE=HORIZONTAL"),
    NULL);
  IupSetAttribute(box, "MARGIN", "10x10");
  IupSetAttribute(box, "GAP", "5");

  dlg = IupDialog(box);
  IupSetAttribute(dlg, "TITLE", "Go To Line");
  IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
  IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_ok);
  IupSetAttributeHandle(dlg, "DEFAULTESC", bt_cancel);
  IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_goto));
```

```
  IupPopup(dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(dlg, "STATUS") == 1)
  {
    int line = IupGetInt(txt, "VALUE");
    int pos;
    IupTextConvertLinColToPos(multitext, line, 0, &pos);
    IupSetInt(multitext, "CARETPOS", pos);
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }

  IupDestroy(dlg);

  return IUP_DEFAULT;
}

int find_next_action_cb(Ihandle* bt_next)
{
  Ihandle* multitext = (Ihandle*)IupGetAttribute(bt_next, "MULTITEXT");
  char* str = IupGetAttribute(multitext, "VALUE");
  int find_pos = IupGetInt(multitext, "FIND_POS");

  Ihandle* txt = IupGetDialogChild(bt_next, "FIND_TEXT");
  char* str_to_find = IupGetAttribute(txt, "VALUE");

  Ihandle* find_case = IupGetDialogChild(bt_next, "FIND_CASE");
  int casesensitive = IupGetInt(find_case, "VALUE");

  int pos = str_find(str + find_pos, str_to_find, casesensitive);
  if (pos >= 0)
    pos += find_pos;
  else if (find_pos > 0)
    pos = str_find(str, str_to_find, casesensitive);  /* try again from the start *

  if (pos >= 0)
  {
    int lin, col,
      end_pos = pos + (int)strlen(str_to_find);

    IupSetInt(multitext, "FIND_POS", end_pos);

    IupSetFocus(multitext);
```

```
    IupSetfAttribute(multitext, "SELECTIONPOS", "%d:%d", pos, end_pos);

    IupTextConvertPosToLinCol(multitext, pos, &lin, &col);
    IupTextConvertLinColToPos(multitext, lin, 0, &pos);  /* position at col=0, just
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }
  else
    IupMessage("Warning", "Text not found.");

  return IUP_DEFAULT;
}

int find_close_action_cb(Ihandle* bt_close)
{
  IupHide(IupGetDialog(bt_close));
  return IUP_DEFAULT;
}

int item_find_action_cb(Ihandle* item_find)
{
  Ihandle* dlg = (Ihandle*)IupGetAttribute(item_find, "FIND_DIALOG");
  if (!dlg)
  {
    Ihandle* multitext = IupGetDialogChild(item_find, "MULTITEXT");
    Ihandle *box, *bt_next, *bt_close, *txt, *find_case;

    txt = IupText(NULL);
    IupSetAttribute(txt, "NAME", "FIND_TEXT");
    IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
    find_case = IupToggle("Case Sensitive", NULL);
    IupSetAttribute(find_case, "NAME", "FIND_CASE");
    bt_next = IupButton("Find Next", NULL);
    IupSetAttribute(bt_next, "PADDING", "10x2");
    IupSetCallback(bt_next, "ACTION", (Icallback)find_next_action_cb);
    bt_close = IupButton("Close", NULL);
    IupSetCallback(bt_close, "ACTION", (Icallback)find_close_action_cb);
    IupSetAttribute(bt_close, "PADDING", "10x2");

    box = IupVbox(
      IupLabel("Find What:"),
      txt,
      find_case,
```

```
      IupSetAttributes(IupHbox(
        IupFill(),
        bt_next,
        bt_close,
        NULL), "NORMALIZESIZE=HORIZONTAL"),
      NULL);
    IupSetAttribute(box, "MARGIN", "10x10");
    IupSetAttribute(box, "GAP", "5");

    dlg = IupDialog(box);
    IupSetAttribute(dlg, "TITLE", "Find");
    IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
    IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_next);
    IupSetAttributeHandle(dlg, "DEFAULTESC", bt_close);
    IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_find));

    /* Save the multiline to acess it from the callbacks */
    IupSetAttribute(dlg, "MULTITEXT", (char*)multitext);

    /* Save the dialog to reuse it */
    IupSetAttribute(item_find, "FIND_DIALOG", (char*)dlg);
  }

  /* centerparent first time, next time reuse the last position */
  IupShowXY(dlg, IUP_CURRENT, IUP_CURRENT);

  return IUP_DEFAULT;
}

int item_font_action_cb(Ihandle* item_font)
{
  Ihandle* multitext = IupGetDialogChild(item_font, "MULTITEXT");
  Ihandle* fontdlg = IupFontDlg();
  char* font = IupGetAttribute(multitext, "FONT");
  IupSetStrAttribute(fontdlg, "VALUE", font);
  IupSetAttributeHandle(fontdlg, "PARENTDIALOG", IupGetDialog(item_font));

  IupPopup(fontdlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(fontdlg, "STATUS") == 1)
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
```

```
    font = IupGetAttribute(fontdlg, "VALUE");
    IupSetStrAttribute(multitext, "FONT", font);

    IupConfigSetVariableStr(config, "MainWindow", "Font", font);
  }

  IupDestroy(fontdlg);
  return IUP_DEFAULT;
}

int item_about_action_cb(void)
{
  IupMessage("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣S
  return IUP_DEFAULT;
}


/********************************** Main ****************************************/


int main(int argc, char **argv)
{
  Ihandle *dlg, *vbox, *multitext, *menu;
  Ihandle *sub_menu_file, *file_menu, *item_exit, *item_open, *item_saveas, *btn_op
  Ihandle *sub_menu_edit, *edit_menu, *item_find, *item_goto, *btn_find;
  Ihandle *sub_menu_format, *format_menu, *item_font;
  Ihandle *sub_menu_help, *help_menu, *item_about;
  Ihandle *lbl_statusbar, *toolbar_hb, *recent_menu;
  Ihandle *config;
  const char* font;

  IupOpen(&argc, &argv);
  IupImageLibOpen();

  config = IupConfig();
  IupSetAttribute(config, "APP_NAME", "simple_notepad");
  IupConfigLoad(config);

  multitext = IupText(NULL);
  IupSetAttribute(multitext, "MULTILINE", "YES");
  IupSetAttribute(multitext, "EXPAND", "YES");
  IupSetAttribute(multitext, "NAME", "MULTITEXT");
```

```
font = IupConfigGetVariableStr(config, "MainWindow", "Font");
if (font)
  IupSetStrAttribute(multitext, "FONT", font);

lbl_statusbar = IupLabel("Lin 1, Col 1");
IupSetAttribute(lbl_statusbar, "NAME", "STATUSBAR");
IupSetAttribute(lbl_statusbar, "EXPAND", "HORIZONTAL");
IupSetAttribute(lbl_statusbar, "PADDING", "10x5");

item_open = IupItem("&Open...\tCtrl+O", NULL);
btn_open = IupButton(NULL, NULL);
IupSetAttribute(btn_open, "IMAGE", "IUP_FileOpen");
IupSetAttribute(btn_open, "FLAT", "Yes");
IupSetAttribute(btn_open, "TIP", "Open (Ctrl+O)");
IupSetAttribute(btn_open, "CANFOCUS", "No");

item_saveas = IupItem("Save &As...\tCtrl+S", NULL);
btn_save = IupButton(NULL, NULL);
IupSetAttribute(btn_save, "IMAGE", "IUP_FileSave");
IupSetAttribute(btn_save, "FLAT", "Yes");
IupSetAttribute(btn_save, "TIP", "Save (Ctrl+S)");
IupSetAttribute(btn_save, "CANFOCUS", "No");

item_exit = IupItem("E&xit", NULL);

item_find = IupItem("&Find...\tCtrl+F", NULL);
btn_find = IupButton(NULL, NULL);
IupSetAttribute(btn_find, "IMAGE", "IUP_EditFind");
IupSetAttribute(btn_find, "FLAT", "Yes");
IupSetAttribute(btn_find, "TIP", "Find (Ctrl+F)");
IupSetAttribute(btn_find, "CANFOCUS", "No");

toolbar_hb = IupHbox(
  btn_open,
  btn_save,
  IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
  btn_find,
  NULL);
IupSetAttribute(toolbar_hb, "MARGIN", "5x5");
IupSetAttribute(toolbar_hb, "GAP", "2");
```

```
  item_goto = IupItem("&Go To...\tCtrl+G", NULL);
  item_font = IupItem("&Font...", NULL);
  item_about = IupItem("&About...", NULL);

  IupSetCallback(item_open, "ACTION", (Icallback)item_open_action_cb);
  IupSetCallback(btn_open, "ACTION", (Icallback)item_open_action_cb);
  IupSetCallback(item_saveas, "ACTION", (Icallback)item_saveas_action_cb);
  IupSetCallback(btn_save, "ACTION", (Icallback)item_saveas_action_cb);
  IupSetCallback(item_exit, "ACTION", (Icallback)item_exit_action_cb);
  IupSetCallback(item_find, "ACTION", (Icallback)item_find_action_cb);
  IupSetCallback(btn_find, "ACTION", (Icallback)item_find_action_cb);
  IupSetCallback(item_goto, "ACTION", (Icallback)item_goto_action_cb);
  IupSetCallback(item_font, "ACTION", (Icallback)item_font_action_cb);
  IupSetCallback(item_about, "ACTION", (Icallback)item_about_action_cb);
  IupSetCallback(multitext, "CARET_CB", (Icallback)multitext_caret_cb);

  recent_menu = IupMenu(NULL);

  file_menu = IupMenu(
    item_open,
    item_saveas,
    IupSeparator(),
    IupSubmenu("Recent &Files", recent_menu),
    item_exit,
    NULL);
  edit_menu = IupMenu(
    item_find,
    item_goto,
    NULL);
  format_menu = IupMenu(
    item_font,
    NULL);
  help_menu = IupMenu(
    item_about,
    NULL);

  sub_menu_file = IupSubmenu("&File", file_menu);
  sub_menu_edit = IupSubmenu("&Edit", edit_menu);
  sub_menu_format = IupSubmenu("F&ormat", format_menu);
  sub_menu_help = IupSubmenu("&Help", help_menu);

  menu = IupMenu(
```

```
    sub_menu_file,
    sub_menu_edit,
    sub_menu_format,
    sub_menu_help,
    NULL);

  vbox = IupVbox(
    toolbar_hb,
    multitext,
    lbl_statusbar,
    NULL);

  dlg = IupDialog(vbox);
  IupSetAttributeHandle(dlg, "MENU", menu);
  IupSetAttribute(dlg, "TITLE", "Simple Notepad");
  IupSetAttribute(dlg, "SIZE", "HALFxHALF");
  IupSetCallback(dlg, "CLOSECB", (Icallback)item_exit_action_cb);

  IupSetAttribute(dlg, "CONFIG", (char*)config);

  /* parent for pre-defined dialogs in closed functions (IupMessage) */
  IupSetAttributeHandle(NULL, "PARENTDIALOG", dlg);

  IupSetCallback(dlg, "K_cO", (Icallback)item_open_action_cb);
  IupSetCallback(dlg, "K_cS", (Icallback)item_saveas_action_cb);
  IupSetCallback(dlg, "K_cF", (Icallback)item_find_action_cb);
  IupSetCallback(dlg, "K_cG", (Icallback)item_goto_action_cb);

  IupConfigRecentInit(config, recent_menu, config_recent_cb, 10);

  IupConfigDialogShow(config, dlg, "MainWindow");

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")
require("iupluaimglib")
```

```
--********************************* Utilities *********************************

function str_find(str, str_to_find, casesensitive, start)
  if (not casesensitive) then
    return str_find(string.lower(str), string.lower(str_to_find), true, start)
  end

  return string.find(str, str_to_find, start, true)
end

function read_file(filename)
  local ifile = io.open(filename, "r")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return nil
  end

  local str = ifile:read("*a")
  if (not str) then
    iup.Message("Error", "Fail when reading from file: " .. filename)
    return nil
  end

  ifile:close()
  return str
end

function write_file(filename, str)
  local ifile = io.open(filename, "w")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return false
  end

  if (not ifile:write(str)) then
    iup.Message("Error", "Fail when writing to file: " .. filename)
  end

  ifile:close()
  return true
```

```
end


--********************************* Main (Part 1/2) ****************************


config = iup.config{}
config.app_name = "simple_notepad"
config:Load()

lbl_statusbar = iup.label{title = "Lin 1, Col 1", expand = "HORIZONTAL", padding =

multitext = iup.text{
  multiline = "YES",
  expand = "YES"
}

font = config:GetVariable("MainWindow", "Font")
if (font) then
  multitext.font = font
end

item_open = iup.item{title = "&Open...\tCtrl+O"}
item_saveas = iup.item{title="Save &As...\tCtrl+S"}
item_font = iup.item{title="&Font..."}
item_about = iup.item{title="&About..."}
item_find = iup.item{title="&Find...\tCtrl+F"}
item_goto = iup.item{title="&Go To..."}
item_exit = iup.item{title="E&xit"}


--********************************* Callbacks **********************************


function config:recent_cb()
  local filename = self.title
  local str = read_file(filename)
  if (str) then
    multitext.value = str
  end
end
```

```
function multitext:caret_cb(lin, col)
  lbl_statusbar.title = "Lin "..lin.." , Col "..col
end

function item_open:action()
  local filedlg = iup.filedlg{
    dialogtype = "OPEN",
    filter = "*.txt",
    filterinfo = "Text Files",
    parentdialog=iup.GetDialog(self)
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    local str = read_file(filename)
    if (str) then
      config:RecentUpdate(filename)
      multitext.value = str
    end
  end
  filedlg:destroy()
end

function item_saveas:action()
  local filedlg = iup.filedlg{
    dialogtype = "SAVE",
    filter = "*.txt",
    filterinfo = "Text Files",
    parentdialog=iup.GetDialog(self)
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    if (write_file(filename, multitext.value)) then
      config:RecentUpdate(filename)
    end
  end
  filedlg:destroy()
```

```
end

function item_exit:action()
  config:DialogClosed(iup.GetDialog(self), "MainWindow")
  config:Save()
  config:destroy()
  return iup.CLOSE
end

function item_goto:action()
  local line_count = multitext.linecount
  local lbl_goto = iup.label{title = "Line Number [1-"..line_count.."]:"}
  local txt_goto = iup.text{mask = iup.MASK_UINT, visiblecolumns = 20} --unsigned i

  local bt_goto_ok = iup.button{title = "OK", text_linecount = 0, padding = "10x2"}
  bt_goto_ok.text_linecount = line_count
  function bt_goto_ok:action()
    local line_count = tonumber(self.text_linecount)
    local line = tonumber(txt_goto.value)
    if (line < 1 or line >= line_count) then
      iup.Message("Error", "Invalid line number.")
      return
    end
    goto_dlg.status = 1
    return iup.CLOSE
  end

  local bt_goto_cancel = iup.button{title = "Cancel", padding = "10x2"}
  function bt_goto_cancel:action()
    goto_dlg.status = 0
    return iup.CLOSE
  end

  local box = iup.vbox{
    lbl_goto,
    txt_goto,
    iup.hbox{
      iup.fill{},
      bt_goto_ok,
      bt_goto_cancel,
      normalizesize="HORIZONTAL",
    },
```

```
    margin = "10x10",
    gap = "5",
  }
  goto_dlg = iup.dialog{
    box,
    title = "Go To Line",
    dialogframe = "Yes",
    defaultenter = bt_goto_ok,
    defaultesc = bt_goto_cancel,
    parentdialog = iup.GetDialog(self)
  }

  goto_dlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(goto_dlg.status) == 1) then
    local line = txt_goto.value
    local pos = iup.TextConvertLinColToPos(multitext, line, 0)
    multitext.caretpos = pos
    multitext.scrolltopos = pos
  end

  goto_dlg:destroy()
end

function item_find:action()
  local find_dlg = self.find_dialog
  if (not find_dlg) then
    local find_txt = iup.text{visiblecolumns = "20"}
    local find_case = iup.toggle{title = "Case Sensitive"}
    local bt_find_next = iup.button{title = "Find Next", padding = "10x2"}
    local bt_find_close = iup.button{title = "Close", padding = "10x2"}

    function bt_find_next:action()
      local find_pos = tonumber(find_dlg.find_pos)
      local str_to_find = find_txt.value

      local casesensitive = (find_case.value == "ON")

      -- test again, because it can be called from the hot key
      if (not str_to_find or str_to_find:len()==0) then
        return
      end
```

```
  if (not find_pos) then
    find_pos = 1
  end

  local str = multitext.value

  local pos, end_pos = str_find(str, str_to_find, casesensitive, find_pos)
  if (not pos) then
    pos, end_pos = str_find(str, str_to_find, casesensitive, 1)  -- try again fr
  end

  if (pos) and (pos > 0) then
    pos = pos - 1
    find_dlg.find_pos = end_pos

    iup.SetFocus(multitext)
    multitext.selectionpos = pos..":"..end_pos

    local lin, col = iup.TextConvertPosToLinCol(multitext, pos)
    local pos = iup.TextConvertLinColToPos(multitext, lin, 0)  -- position at c
    multitext.scrolltopos = pos
  else
    find_dlg.find_pos = nil
    iup.Message("Warning", "Text not found.")
  end
end

function bt_find_close:action()
  iup.Hide(iup.GetDialog(self))  -- do not destroy, just hide
end

box = iup.vbox{
  iup.label{title = "Find What:"},
  find_txt,
  find_case,
  iup.hbox{
    iup.fill{},
    bt_find_next,
    bt_find_close,
    normalizesize="HORIZONTAL",
  },
```

```
      margin = "10x10",
      gap = "5",
    }

    find_dlg = iup.dialog{
      box,
      title = "Find",
      dialogframe = "Yes",
      defaultenter = bt_next,
      defaultesc = bt_close,
      parentdialog = iup.GetDialog(self)
      }

    -- Save the dialog to reuse it
    self.find_dialog = find_dlg -- from the main dialog */
  end

  -- centerparent first time, next time reuse the last position
  find_dlg:showxy(iup.CURRENT, iup.CURRENT)
end

function item_font:action()
  local font = multitext.font
  local fontdlg = iup.fontdlg{value = font, parentdialog=iup.GetDialog(self)}

  fontdlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(fontdlg.status) == 1) then
    multitext.font = fontdlg.value
    config:SetVariable("MainWindow", "Font", fontdlg.value)
  end

  fontdlg:destroy()
end

function item_about:action()
  iup.Message("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣
end


--********************************* Main (Part 2/2) ****************************
```

```
recent_menu = iup.menu{}

file_menu = iup.menu{
  item_open,
  item_saveas,
  iup.separator{},
  iup.submenu{title="Recent␣&Files", recent_menu},
  item_exit
  }
edit_menu = iup.menu{item_find, item_goto}
format_menu = iup.menu{item_font}
help_menu = iup.menu{item_about}
sub_menu_file = iup.submenu{file_menu, title = "&File"}
sub_menu_edit = iup.submenu{edit_menu, title = "&Edit"}
sub_menu_format = iup.submenu{format_menu, title = "F&ormat"}
sub_menu_help = iup.submenu{help_menu, title = "&Help"}

menu = iup.menu{
  sub_menu_file,
  sub_menu_edit,
  sub_menu_format,
  sub_menu_help,
  }

btn_open = iup.button{image = "IUP_FileOpen", flat = "Yes", action = item_open.actio
btn_save = iup.button{image = "IUP_FileSave", flat = "Yes", action = item_saveas.ac
btn_find = iup.button{image = "IUP_EditFind", flat = "Yes", action = item_find.actio

toolbar_hb = iup.hbox{
  btn_open,
  btn_save,
  iup.label{separator="VERTICAL"},
  btn_find,
  margin = "5x5",
  gap = 2,
  }

vbox = iup.vbox{
  toolbar_hb,
  multitext,
  lbl_statusbar,
```

```
}

dlg = iup.dialog{
  vbox,
  title = "Simple Notepad",
  size = "HALFxHALF",
  menu = menu,
  close_cb = item_exit.action,
}

function dlg:k_any(c)
  if (c == iup.K_cO) then
    item_open:action()
  elseif (c == iup.K_cS) then
    item_saveas:action()
  elseif (c == iup.K_cF) then
    item_find:action()
  elseif (c == iup.K_cG) then
    item_goto:action()
  end
end

config:RecentInit(recent_menu, 10)

-- parent for pre-defined dialogs in closed functions (IupMessage)
iup.SetGlobal("PARENTDIALOG", iup.SetHandleName(dlg))

config:DialogShow(dlg, "MainWindow")

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

可以看到，我们包含了 *iup_config.h* 头文件。我们现在开始分析主函数。在创建 *IupConfig* 句柄之后，我们设置了它的 *APP_NAME* 属性。这个属性定义了我们的配置文件名称。在 Unix 下，配置文件名会是 *<HOME>/.<APP_NAME>*，其中 *<HOME>* 是系统的 *HOME* 环境变量内容，*<APP_NAME>* 是 *APP_NAME* 属性值。在 Windows 下，文件名会是 *<HOMEDRIVE><HOMEPATH>*
<APP_NAME>.cfg，其中 *HOMEDRIVE* 和 *HOMEPATH* 都是从环境变量获取的。

之后，我们调用 *IupConfigLoad* 函数载入配置文件。这个函数和 *IupConfigSave* 函数是对应的，我们通过它们来实现程序数据的持久化。

接着，我们创建了 *recent_menu* 菜单项目来存放我们的最近文件菜单项目。它和其它菜单的创建没有区别，除了我们没有往里面添加菜单项。它的菜单项目将由后面的一个函数提供。我们在 *recent_menu* 菜单和 *item_exit* 菜单项目之间放置了一个 *IupSeparator* 元素来进行划分。

在菜单创建之后,我们调用了 *IupConfigRecentInit* 函数。这个函数根据配置文件对 *recent_menu* 菜单项目进行初始化。用户点击最近文件列表中的项目后，*item_recent_cb* 回调函数会被调用。这个函数也定义了需要显示和存储的最近文件的数量。在本例中，我们存储了 10 个文件。另外需要注意，*item_open_cb* 和 *item_saveas_cb* 回调函数都会改变最近文件列表。所以记得调用 *IupConfigRecentUpdate* 函数来维护我们的最近文件列表就很有必要。

下一行，我们调用了 *IupConfigDialogShow* 函数。这个函数除了显示窗口，还会尝试使用存储在配置文件中的上一次窗口位置和窗口大小等信息。它可以被任意的程序窗口所使用，只需要为窗口使用不同的名称即可。

IupConfigDialogClosed 函数被用来存储即将关闭的窗口的位置以及大小信息。通常，它在窗口的 *CLOSE_CB* 回调或窗口被隐藏时被手动调用。当用户点击窗口的关闭按钮，*CLOSE_CB* 回调就会被调用。在我们的程序中，使用 *item_exit_action_cb* 回调函数来处理程序退出，所以我们同时也可以把它作为 *CLOSE_CB* 的回调函数，然后在其中调用 *IupConfigDialogClosed* 函数。最后，我们调用 *IupConfigSave* 函数来保存我们的配置到配置文件。之后，我们使用 *IupDestroy* 函数销毁配置元素的句柄。

上面是主要的部分，现在让我们把注意力转向 *item_recent_cb* 回调函数。这个回调函数对用户点击最近文件列表后进行处理。在里面，我们通过窗口的自定义属性来恢复我们的配置句柄。然后，我们通过 *TITLE* 属性获取文件名，然后执行和 *item_open_cb* 一样的操作。

3.8 剪贴板支持

接下来，我们会看到一些关于复制，剪切，粘贴，删除和全选回调函数，它们和 *Edit* 的菜单项目相关联。除此之外，还有一个回调函数对这些菜单项目的激活状态进行管理。它们的代码都非常短。

item_copy_action_cb，*item_paste_action_cb* 和 *item_cut_action_cb* 使用 *IupClipboard* 元素来进行对剪贴板的访问。*IupClipboard* 在不使用后应该调用 *IupDestroy* 函数来销毁它。我们可以在整个程序中使用一个 *IupClipboard* 元素，也可以在需要使用它的地方每次创建它，使用它，然后销毁它。*item_copy_action_cb* 回调函数会获取文本框的 *SELECTEDTEXT* 属性，然后把获取的数据设置为剪贴板的 *TEXT* 属性值。*item_paste_action_cb* 回调函数获取剪贴板的 *TEXT* 属性值，然后使用 *INSERT* 属性将它插入到文本框的文本中，插入的位置是当前文本框的光标位置。*item_cut_action_cb* 回调函数的代码和复制几乎是相同的，除了它在之后把 *SELECTEDTEXT* 的值设置为空字符串。从而删除文本框中被选择的字符串。*item_delete_action_cb* 回调函数的内容和剪贴类似，但不使用剪贴板。*item_select_all_cb* 回调函数设置 *SELECTION* 属性为 *ALL* 来选择文本框中的所有文本。

我们创建了一个新的回调函数来初始化我们新的菜单项。*edit_menu_open_cb* 回调函数和 *edit_menu* 的 *OPEN_CB* 回调相关联。它将通过一些信息设置剪贴，粘贴，复制，和删除菜单项目是否被激活。首先，我们需要获得这些元素的句柄。我们通过 *IupGetDialogChild* 函数就可以通过元素的 *NAME* 属性来获取元素的句柄。接着，我们使用 *IupGetInt* 函数检查剪贴板的 *TEXTAVAIABLE* 属性，来确定剪贴板是否有文本存在。如果这个函数返回 0，说明剪贴板中没有文本，或者说就是没有东西可以去粘贴。这时，我们应该通过 *paste* 菜单元素的 *ACTIVE* 属性来禁用 *paste* 菜单项目。在可以使用剪贴板时，设置它的 *ACTIVE* 属性为 *YES*，再不能使用时设置

它的 *ACTION* 属性为 *NO*。其它菜单项目的激活与否与此类次，不同的地方是，检查的属性可能是 *SELECTEDTEXT*，如果它为空，我们就要禁用 *cut* 菜单，*copy* 菜单和 *delete* 菜单，否则的话，我们就应该激活它们。

C语言

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <iup.h>
#include <iup_config.h>


/********************************* Utilities *************************************


int str_compare(const char *l, const char *r, int casesensitive)
{
  if (!l || !r)
    return 0;

  while (*l && *r)
  {
    int diff;
    char l_char = *l,
      r_char = *r;

    /* compute the difference of both characters */
    if (casesensitive)
      diff = l_char - r_char;
    else
      diff = tolower((int)l_char) - tolower((int)r_char);

    /* if they differ we have a result */
    if (diff != 0)
      return 0;

    /* otherwise process the next characters */
    ++l;
    ++r;
  }

  /* check also for terminator */
```

```
  if (*l == *r)
    return 1;

  if (*r == 0)
    return 1;  /* if second string is at terminator, then it is partially equal */

  return 0;
}

int str_find(const char *str, const char *str_to_find, int casesensitive)
{
  int i, str_len, str_to_find_len, count;

  if (!str || str[0] == 0 || !str_to_find || str_to_find[0] == 0)
    return -1;

  str_len = (int)strlen(str);
  str_to_find_len = (int)strlen(str_to_find);
  count = str_len - str_to_find_len;
  if (count < 0)
    return -1;

  count++;

  for (i = 0; i<count; i++)
  {
    if (str_compare(str, str_to_find, casesensitive))
      return i;

    str++;
  }

  return -1;
}

char* read_file(const char* filename)
{
  int size;
  char* str;
  FILE* file = fopen(filename, "rb");
  if (!file)
  {
```

```
    IupMessagef("Error", "Can't open file: %s", filename);
    return NULL;
  }

  /* calculate file size */
  fseek(file, 0, SEEK_END);
  size = ftell(file);
  fseek(file, 0, SEEK_SET);

  /* allocate memory for the file contents + nul terminator */
  str = malloc(size + 1);
  /* read all data at once */
  fread(str, size, 1, file);
  /* set the nul terminator */
  str[size] = 0;

  if (ferror(file))
    IupMessagef("Error", "Fail when reading from file: %s", filename);

  fclose(file);
  return str;
}

int write_file(const char* filename, const char* str, int count)
{
  FILE* file = fopen(filename, "w");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return 0;
  }

  fwrite(str, 1, count, file);

  if (ferror(file))
    IupMessagef("Error", "Fail when writing to file: %s", filename);

  fclose(file);
  return 1;
}
```

```
/********************************* Callbacks *************************************


int edit_menu_open_cb(Ihandle* ih)
{
  Ihandle *clipboard = IupClipboard();
  Ihandle *item_paste = IupGetDialogChild(ih, "ITEM_PASTE");
  Ihandle *item_cut = IupGetDialogChild(ih, "ITEM_CUT");
  Ihandle *item_delete = IupGetDialogChild(ih, "ITEM_DELETE");
  Ihandle *item_copy = IupGetDialogChild(ih, "ITEM_COPY");
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  if (!IupGetInt(clipboard, "TEXTAVAILABLE"))
    IupSetAttribute(item_paste, "ACTIVE", "NO");
  else
    IupSetAttribute(item_paste, "ACTIVE", "YES");

  if (!IupGetAttribute(multitext, "SELECTEDTEXT"))
  {
    IupSetAttribute(item_cut, "ACTIVE", "NO");
    IupSetAttribute(item_delete, "ACTIVE", "NO");
    IupSetAttribute(item_copy, "ACTIVE", "NO");
  }
  else
  {
    IupSetAttribute(item_cut, "ACTIVE", "YES");
    IupSetAttribute(item_delete, "ACTIVE", "YES");
    IupSetAttribute(item_copy, "ACTIVE", "YES");
  }

  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int config_recent_cb(Ihandle* ih)
{
  char* filename = IupGetAttribute(ih, "TITLE");
  char* str = read_file(filename);
  if (str)
  {
    Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");
    IupSetStrAttribute(multitext, "VALUE", str);
```

```
    free(str);
  }
  return IUP_DEFAULT;
}

int multitext_caret_cb(Ihandle *ih, int lin, int col)
{
  Ihandle *lbl_statusbar = IupGetDialogChild(ih, "STATUSBAR");
  IupSetfAttribute(lbl_statusbar, "TITLE", "Lin %d, Col %d", lin, col);
  return IUP_DEFAULT;
}

int item_open_action_cb(Ihandle* item_open)
{
  Ihandle* multitext = IupGetDialogChild(item_open, "MULTITEXT");
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "OPEN");
  IupSetAttribute(filedlg, "EXTFILTER", "Text Files|*.txt|All Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_open));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    char* str = read_file(filename);
    if (str)
    {
      Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
      IupConfigRecentUpdate(config, filename);

      IupSetStrAttribute(multitext, "VALUE", str);
      free(str);
    }
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_saveas_action_cb(Ihandle* item_saveas)
{
```

```
  Ihandle* multitext = IupGetDialogChild(item_saveas, "MULTITEXT");
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "SAVE");
  IupSetAttribute(filedlg, "EXTFILTER", "Text␣Files|*.txt|All␣Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_saveas));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
    char* filename = IupGetAttribute(filedlg, "VALUE");
    char* str = IupGetAttribute(multitext, "VALUE");
    int count = IupGetInt(multitext, "COUNT");
    if (write_file(filename, str, count))
      IupConfigRecentUpdate(config, filename);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_exit_action_cb(Ihandle* item_exit)
{
  Ihandle* dlg = IupGetDialog(item_exit);
  Ihandle* config = (Ihandle*)IupGetAttribute(dlg, "CONFIG");
  IupConfigDialogClosed(config, dlg, "MainWindow");
  IupConfigSave(config);
  IupDestroy(config);
  return IUP_CLOSE;
}

int goto_ok_action_cb(Ihandle* bt_ok)
{
  int line_count = IupGetInt(bt_ok, "TEXT_LINECOUNT");
  Ihandle* txt = IupGetDialogChild(bt_ok, "LINE_TEXT");
  int line = IupGetInt(txt, "VALUE");
  if (line < 1 || line >= line_count)
  {
    IupMessage("Error", "Invalid␣line␣number.");
    return IUP_DEFAULT;
  }
```

```
  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "1");
  return IUP_CLOSE;
}

int goto_cancel_action_cb(Ihandle* bt_ok)
{
  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "0");
  return IUP_CLOSE;
}

int item_goto_action_cb(Ihandle* item_goto)
{
  Ihandle* multitext = IupGetDialogChild(item_goto, "MULTITEXT");
  Ihandle *dlg, *box, *bt_ok, *bt_cancel, *txt, *lbl;

  int line_count = IupGetInt(multitext, "LINECOUNT");

  lbl = IupLabel(NULL);
  IupSetfAttribute(lbl, "TITLE", "Line Number [1-%d]:", line_count);
  txt = IupText(NULL);
  IupSetAttribute(txt, "MASK", IUP_MASK_UINT);  /* unsigned integer numbers only */
  IupSetAttribute(txt, "NAME", "LINE_TEXT");
  IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
  bt_ok = IupButton("OK", NULL);
  IupSetInt(bt_ok, "TEXT_LINECOUNT", line_count);
  IupSetAttribute(bt_ok, "PADDING", "10x2");
  IupSetCallback(bt_ok, "ACTION", (Icallback)goto_ok_action_cb);
  bt_cancel = IupButton("Cancel", NULL);
  IupSetCallback(bt_cancel, "ACTION", (Icallback)goto_cancel_action_cb);
  IupSetAttribute(bt_cancel, "PADDING", "10x2");

  box = IupVbox(
    lbl,
    txt,
    IupSetAttributes(IupHbox(
      IupFill(),
      bt_ok,
      bt_cancel,
      NULL), "NORMALIZESIZE=HORIZONTAL"),
    NULL);
  IupSetAttribute(box, "MARGIN", "10x10");
```

```
  IupSetAttribute(box, "GAP", "5");

  dlg = IupDialog(box);
  IupSetAttribute(dlg, "TITLE", "Go To Line");
  IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
  IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_ok);
  IupSetAttributeHandle(dlg, "DEFAULTESC", bt_cancel);
  IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_goto));

  IupPopup(dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(dlg, "STATUS") == 1)
  {
    int line = IupGetInt(txt, "VALUE");
    int pos;
    IupTextConvertLinColToPos(multitext, line, 0, &pos);
    IupSetInt(multitext, "CARETPOS", pos);
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }

  IupDestroy(dlg);

  return IUP_DEFAULT;
}

int find_next_action_cb(Ihandle* bt_next)
{
  Ihandle* multitext = (Ihandle*)IupGetAttribute(bt_next, "MULTITEXT");
  char* str = IupGetAttribute(multitext, "VALUE");
  int find_pos = IupGetInt(multitext, "FIND_POS");

  Ihandle* txt = IupGetDialogChild(bt_next, "FIND_TEXT");
  char* str_to_find = IupGetAttribute(txt, "VALUE");

  Ihandle* find_case = IupGetDialogChild(bt_next, "FIND_CASE");
  int casesensitive = IupGetInt(find_case, "VALUE");

  int pos = str_find(str + find_pos, str_to_find, casesensitive);
  if (pos >= 0)
    pos += find_pos;
  else if (find_pos > 0)
    pos = str_find(str, str_to_find, casesensitive);  /* try again from the start *
```

```
  if (pos >= 0)
  {
    int lin, col,
      end_pos = pos + (int)strlen(str_to_find);

    IupSetInt(multitext, "FIND_POS", end_pos);

    IupSetFocus(multitext);
    IupSetfAttribute(multitext, "SELECTIONPOS", "%d:%d", pos, end_pos);

    IupTextConvertPosToLinCol(multitext, pos, &lin, &col);
    IupTextConvertLinColToPos(multitext, lin, 0, &pos);  /* position at col=0, just
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }
  else
    IupMessage("Warning", "Text not found.");

  return IUP_DEFAULT;
}

int find_close_action_cb(Ihandle* bt_close)
{
  IupHide(IupGetDialog(bt_close));
  return IUP_DEFAULT;
}

int item_find_action_cb(Ihandle* item_find)
{
  Ihandle* dlg = (Ihandle*)IupGetAttribute(item_find, "FIND_DIALOG");
  if (!dlg)
  {
    Ihandle* multitext = IupGetDialogChild(item_find, "MULTITEXT");
    Ihandle *box, *bt_next, *bt_close, *txt, *find_case;

    txt = IupText(NULL);
    IupSetAttribute(txt, "NAME", "FIND_TEXT");
    IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
    find_case = IupToggle("Case Sensitive", NULL);
    IupSetAttribute(find_case, "NAME", "FIND_CASE");
    bt_next = IupButton("Find Next", NULL);
    IupSetAttribute(bt_next, "PADDING", "10x2");
```

```
    IupSetCallback(bt_next, "ACTION", (Icallback)find_next_action_cb);
    bt_close = IupButton("Close", NULL);
    IupSetCallback(bt_close, "ACTION", (Icallback)find_close_action_cb);
    IupSetAttribute(bt_close, "PADDING", "10x2");

    box = IupVbox(
      IupLabel("Find␣What:"),
      txt,
      find_case,
      IupSetAttributes(IupHbox(
        IupFill(),
        bt_next,
        bt_close,
        NULL), "NORMALIZESIZE=HORIZONTAL"),
      NULL);
    IupSetAttribute(box, "MARGIN", "10x10");
    IupSetAttribute(box, "GAP", "5");

    dlg = IupDialog(box);
    IupSetAttribute(dlg, "TITLE", "Find");
    IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
    IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_next);
    IupSetAttributeHandle(dlg, "DEFAULTESC", bt_close);
    IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_find));

    /* Save the multiline to acess it from the callbacks */
    IupSetAttribute(dlg, "MULTITEXT", (char*)multitext);

    /* Save the dialog to reuse it */
    IupSetAttribute(item_find, "FIND_DIALOG", (char*)dlg);
  }

  /* centerparent first time, next time reuse the last position */
  IupShowXY(dlg, IUP_CURRENT, IUP_CURRENT);

  return IUP_DEFAULT;
}

int item_copy_action_cb(Ihandle* item_copy)
{
  Ihandle* multitext = IupGetDialogChild(item_copy, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
```

```
  IupSetAttribute(clipboard, "TEXT", IupGetAttribute(multitext, "SELECTEDTEXT"));
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_paste_action_cb(Ihandle* item_paste)
{
  Ihandle* multitext = IupGetDialogChild(item_paste, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(multitext, "INSERT", IupGetAttribute(clipboard, "TEXT"));
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_cut_action_cb(Ihandle* item_cut)
{
  Ihandle* multitext = IupGetDialogChild(item_cut, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(clipboard, "TEXT", IupGetAttribute(multitext, "SELECTEDTEXT"));
  IupSetAttribute(multitext, "SELECTEDTEXT", "");
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_delete_action_cb(Ihandle* item_delete)
{
  Ihandle* multitext = IupGetDialogChild(item_delete, "MULTITEXT");
  IupSetAttribute(multitext, "SELECTEDTEXT", "");
  return IUP_DEFAULT;
}

int item_select_all_action_cb(Ihandle* item_select_all)
{
  Ihandle* multitext = IupGetDialogChild(item_select_all, "MULTITEXT");
  IupSetFocus(multitext);
  IupSetAttribute(multitext, "SELECTION", "ALL");
  return IUP_DEFAULT;
}

int item_font_action_cb(Ihandle* item_font)
{
  Ihandle* multitext = IupGetDialogChild(item_font, "MULTITEXT");
```

```
  Ihandle* fontdlg = IupFontDlg();
  char* font = IupGetAttribute(multitext, "FONT");
  IupSetStrAttribute(fontdlg, "VALUE", font);
  IupSetAttributeHandle(fontdlg, "PARENTDIALOG", IupGetDialog(item_font));

  IupPopup(fontdlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(fontdlg, "STATUS") == 1)
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
    font = IupGetAttribute(fontdlg, "VALUE");
    IupSetStrAttribute(multitext, "FONT", font);

    IupConfigSetVariableStr(config, "MainWindow", "Font", font);
  }

  IupDestroy(fontdlg);
  return IUP_DEFAULT;
}

int item_about_action_cb(void)
{
  IupMessage("About", "   Simple Notepad\n\nAutors:\n   Gustavo Lyrio\n   Antonio S
  return IUP_DEFAULT;
}


/********************************* Main ****************************************/


int main(int argc, char **argv)
{
  Ihandle *dlg, *vbox, *multitext, *menu;
  Ihandle *sub_menu_file, *file_menu, *item_exit, *item_open, *item_saveas, *btn_op
  Ihandle *sub_menu_edit, *edit_menu, *item_find, *item_goto, *btn_find, *item_copy
  Ihandle *sub_menu_format, *format_menu, *item_font;
  Ihandle *sub_menu_help, *help_menu, *item_about;
  Ihandle *lbl_statusbar, *toolbar_hb, *recent_menu;
  Ihandle *config;
  const char* font;

  IupOpen(&argc, &argv);
```

```
  IupImageLibOpen();

  config = IupConfig();
  IupSetAttribute(config, "APP_NAME", "simple_notepad");
  IupConfigLoad(config);

  multitext = IupText(NULL);
  IupSetAttribute(multitext, "MULTILINE", "YES");
  IupSetAttribute(multitext, "EXPAND", "YES");
  IupSetAttribute(multitext, "NAME", "MULTITEXT");

  font = IupConfigGetVariableStr(config, "MainWindow", "Font");
  if (font)
    IupSetStrAttribute(multitext, "FONT", font);

  lbl_statusbar = IupLabel("Lin 1, Col 1");
  IupSetAttribute(lbl_statusbar, "NAME", "STATUSBAR");
  IupSetAttribute(lbl_statusbar, "EXPAND", "HORIZONTAL");
  IupSetAttribute(lbl_statusbar, "PADDING", "10x5");

  item_open = IupItem("&Open...\tCtrl+O", NULL);
  btn_open = IupButton(NULL, NULL);
  IupSetAttribute(btn_open, "IMAGE", "IUP_FileOpen");
  IupSetAttribute(btn_open, "FLAT", "Yes");
  IupSetAttribute(btn_open, "TIP", "Open (Ctrl+O)");
  IupSetAttribute(btn_open, "CANFOCUS", "No");

  item_saveas = IupItem("Save &As...\tCtrl+S", NULL);
  btn_save = IupButton(NULL, NULL);
  IupSetAttribute(btn_save, "IMAGE", "IUP_FileSave");
  IupSetAttribute(btn_save, "FLAT", "Yes");
  IupSetAttribute(btn_save, "TIP", "Save (Ctrl+S)");
  IupSetAttribute(btn_save, "CANFOCUS", "No");

  item_exit = IupItem("E&xit", NULL);

  item_find = IupItem("&Find...\tCtrl+F", NULL);
  btn_find = IupButton(NULL, NULL);
  IupSetAttribute(btn_find, "IMAGE", "IUP_EditFind");
  IupSetAttribute(btn_find, "FLAT", "Yes");
  IupSetAttribute(btn_find, "TIP", "Find (Ctrl+F)");
  IupSetAttribute(btn_find, "CANFOCUS", "No");
```

```
item_copy = IupItem ("Copy\tCtrl+C", NULL);
IupSetAttribute(item_copy, "NAME", "ITEM_COPY");
item_paste = IupItem ("Paste\tCtrl+V", NULL);
IupSetAttribute(item_paste, "NAME", "ITEM_PASTE");
item_cut = IupItem ("Cut\tCtrl+X", NULL);
IupSetAttribute(item_cut, "NAME", "ITEM_CUT");
item_delete = IupItem ("Delete\tDel", NULL);
IupSetAttribute(item_delete, "NAME", "ITEM_DELETE");
item_select_all = IupItem ("Select All\tCtrl+A", NULL);

toolbar_hb = IupHbox(
  btn_open,
  btn_save,
  IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
  btn_find,
  NULL);
IupSetAttribute(toolbar_hb, "MARGIN", "5x5");
IupSetAttribute(toolbar_hb, "GAP", "2");

item_goto = IupItem("&Go To...\tCtrl+G", NULL);
item_font = IupItem("&Font...", NULL);
item_about = IupItem("&About...", NULL);

IupSetCallback(item_open, "ACTION", (Icallback)item_open_action_cb);
IupSetCallback(btn_open, "ACTION", (Icallback)item_open_action_cb);
IupSetCallback(item_saveas, "ACTION", (Icallback)item_saveas_action_cb);
IupSetCallback(btn_save, "ACTION", (Icallback)item_saveas_action_cb);
IupSetCallback(item_exit, "ACTION", (Icallback)item_exit_action_cb);
IupSetCallback(item_find, "ACTION", (Icallback)item_find_action_cb);
IupSetCallback(btn_find, "ACTION", (Icallback)item_find_action_cb);
IupSetCallback(item_goto, "ACTION", (Icallback)item_goto_action_cb);
IupSetCallback(item_font, "ACTION", (Icallback)item_font_action_cb);
IupSetCallback(item_about, "ACTION", (Icallback)item_about_action_cb);
IupSetCallback(multitext, "CARET_CB", (Icallback)multitext_caret_cb);
IupSetCallback(item_copy, "ACTION", (Icallback)item_copy_action_cb);
IupSetCallback(item_paste, "ACTION", (Icallback)item_paste_action_cb);
IupSetCallback(item_cut, "ACTION", (Icallback)item_cut_action_cb);
IupSetCallback(item_delete, "ACTION", (Icallback)item_delete_action_cb);
IupSetCallback(item_select_all, "ACTION", (Icallback)item_select_all_action_cb);

recent_menu = IupMenu(NULL);
```

```
file_menu = IupMenu(
  item_open,
  item_saveas,
  IupSeparator(),
  IupSubmenu("Recent &Files", recent_menu),
  item_exit,
  NULL);
edit_menu = IupMenu(
  item_cut,
  item_copy,
  item_paste,
  item_delete,
  IupSeparator(),
  item_find,
  item_goto,
  IupSeparator(),
  item_select_all,
  NULL);
format_menu = IupMenu(
  item_font,
  NULL);
help_menu = IupMenu(
  item_about,
  NULL);

sub_menu_file = IupSubmenu("&File", file_menu);
sub_menu_edit = IupSubmenu("&Edit", edit_menu);
sub_menu_format = IupSubmenu("F&ormat", format_menu);
sub_menu_help = IupSubmenu("&Help", help_menu);

menu = IupMenu(
  sub_menu_file,
  sub_menu_edit,
  sub_menu_format,
  sub_menu_help,
  NULL);

vbox = IupVbox(
  toolbar_hb,
  multitext,
  lbl_statusbar,
```

```
    NULL);

  dlg = IupDialog(vbox);
  IupSetAttributeHandle(dlg, "MENU", menu);
  IupSetAttribute(dlg, "TITLE", "Simple Notepad");
  IupSetAttribute(dlg, "SIZE", "HALFxHALF");
  IupSetCallback(dlg, "CLOSECB", (Icallback)item_exit_action_cb);
  IupSetCallback(edit_menu, "OPEN_CB", (Icallback)edit_menu_open_cb);

  IupSetAttribute(dlg, "CONFIG", (char*)config);

  /* parent for pre-defined dialogs in closed functions (IupMessage) */
  IupSetAttributeHandle(NULL, "PARENTDIALOG", dlg);

  IupSetCallback(dlg, "K_cO", (Icallback)item_open_action_cb);
  IupSetCallback(dlg, "K_cS", (Icallback)item_saveas_action_cb);
  IupSetCallback(dlg, "K_cF", (Icallback)item_find_action_cb);
  IupSetCallback(dlg, "K_cG", (Icallback)item_goto_action_cb);

  IupConfigRecentInit(config, recent_menu, config_recent_cb, 10);

  IupConfigDialogShow(config, dlg, "MainWindow");

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")
require("iupluaimglib")


--********************************** Utilities *************************************


function str_find(str, str_to_find, casesensitive, start)
  if (not casesensitive) then
    return str_find(string.lower(str), string.lower(str_to_find), true, start)
  end
```

```
  return string.find(str, str_to_find, start, true)
end

function read_file(filename)
  local ifile = io.open(filename, "r")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return nil
  end

  local str = ifile:read("*a")
  if (not str) then
    iup.Message("Error", "Fail when reading from file: " .. filename)
    return nil
  end

  ifile:close()
  return str
end

function write_file(filename, str)
  local ifile = io.open(filename, "w")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return false
  end

  if (not ifile:write(str)) then
    iup.Message("Error", "Fail when writing to file: " .. filename)
  end

  ifile:close()
  return true
end


--******************************** Main (Part 1/2) ****************************


config = iup.config{}
config.app_name = "simple_notepad"
config:Load()
```

```
lbl_statusbar = iup.label{title = "Lin 1, Col 1", expand = "HORIZONTAL", padding =

multitext = iup.text{
  multiline = "YES",
  expand = "YES"
}

font = config:GetVariable("MainWindow", "Font")
if (font) then
  multitext.font = font
end

item_open = iup.item{title = "&Open...\tCtrl+O"}
item_saveas = iup.item{title="Save &As...\tCtrl+S"}
item_font = iup.item{title="&Font..."}
item_about = iup.item{title="&About..."}
item_find = iup.item{title="&Find...\tCtrl+F"}
item_goto = iup.item{title="&Go To..."}
item_copy = iup.item{title="&Copy\tCtrl+C"}
item_paste = iup.item{title="&Paste\tCtrl+V"}
item_cut = iup.item{title="Cu&t\tCtrl+X"}
item_delete = iup.item{title="&Delete\tDel"}
item_select_all = iup.item{title="Select &All\tCtrl+A"}
item_exit = iup.item{title="E&xit"}


--********************************* Callbacks *********************************


function config:recent_cb()
  local filename = self.title
  local str = read_file(filename)
  if (str) then
    multitext.value = str
  end
end

function multitext:caret_cb(lin, col)
  lbl_statusbar.title = "Lin "..lin.."," Col "..col
end
```

```
function item_open:action()
  local filedlg = iup.filedlg{
    dialogtype = "OPEN",
    filter = "*.txt",
    filterinfo = "Text Files",
    parentdialog=iup.GetDialog(self)
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    local str = read_file(filename)
    if (str) then
      config:RecentUpdate(filename)
      multitext.value = str
    end
  end

  filedlg:destroy()
end

function item_saveas:action()
  local filedlg = iup.filedlg{
    dialogtype = "SAVE",
    filter = "*.txt",
    filterinfo = "Text Files",
    parentdialog = iup.GetDialog(self),
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    if (write_file(filename, multitext.value)) then
      config:RecentUpdate(filename)
    end
  end

  filedlg:destroy()
end
```

```
function item_exit:action()
  config:DialogClosed(iup.GetDialog(self), "MainWindow")
  config:Save()
  config:destroy()
  return iup.CLOSE
end

function item_goto:action()
  local line_count = multitext.linecount
  local lbl_goto = iup.label{title = "Line Number [1-"..line_count.."]:"}
  local txt_goto = iup.text{mask = iup.MASK_UINT, visiblecolumns = 20} --unsigned i

  local bt_goto_ok = iup.button{title = "OK", text_linecount = 0, padding = "10x2"}
  bt_goto_ok.text_linecount = line_count
  function bt_goto_ok:action()
    local line_count = tonumber(self.text_linecount)
    local line = tonumber(txt_goto.value)
    if (line < 1 or line >= line_count) then
      iup.Message("Error", "Invalid line number.")
      return
    end
    goto_dlg.status = 1
    return iup.CLOSE
  end

  local bt_goto_cancel = iup.button{title = "Cancel", padding = "10x2"}
  function bt_goto_cancel:action()
    goto_dlg.status = 0
    return iup.CLOSE
  end

  local box = iup.vbox{
    lbl_goto,
    txt_goto,
    iup.hbox{
      iup.fill{},
      bt_goto_ok,
      bt_goto_cancel,
      normalizesize="HORIZONTAL",
    },
    margin = "10x10",
    gap = "5",
```

```
  }
  goto_dlg = iup.dialog{
    box,
    title = "Go To Line",
    dialogframe = "Yes",
    defaultenter = bt_goto_ok,
    defaultesc = bt_goto_cancel,
    parentdialog = iup.GetDialog(self)
  }

  goto_dlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(goto_dlg.status) == 1) then
    local line = txt_goto.value
    local pos = iup.TextConvertLinColToPos(multitext, line, 0)
    multitext.caretpos = pos
    multitext.scrolltopos = pos
  end

  goto_dlg:destroy()
end

function item_find:action()
  local find_dlg = self.find_dialog
  if (not find_dlg) then
    local find_txt = iup.text{visiblecolumns = "20"}
    local find_case = iup.toggle{title = "Case Sensitive"}
    local bt_find_next = iup.button{title = "Find Next", padding = "10x2"}
    local bt_find_close = iup.button{title = "Close", padding = "10x2"}

    function bt_find_next:action()
      local find_pos = tonumber(find_dlg.find_pos)
      local str_to_find = find_txt.value

      local casesensitive = (find_case.value == "ON")

      -- test again, because it can be called from the hot key
      if (not str_to_find or str_to_find:len()==0) then
        return
      end

      if (not find_pos) then
```

```
    find_pos = 1
  end

  local str = multitext.value

  local pos, end_pos = str_find(str, str_to_find, casesensitive, find_pos)
  if (not pos) then
    pos, end_pos = str_find(str, str_to_find, casesensitive, 1)  -- try again fr
  end

  if (pos) and (pos > 0) then
    pos = pos - 1
    find_dlg.find_pos = end_pos

    iup.SetFocus(multitext)
    multitext.selectionpos = pos..":"..end_pos

    local lin, col = iup.TextConvertPosToLinCol(multitext, pos)
    local pos = iup.TextConvertLinColToPos(multitext, lin, 0)  -- position at c
    multitext.scrolltopos = pos
  else
    find_dlg.find_pos = nil
    iup.Message("Warning", "Text␣not␣found.")
  end
end

function bt_find_close:action()
  iup.Hide(iup.GetDialog(self))  -- do not destroy, just hide
end

box = iup.vbox{
  iup.label{title = "Find␣What:"},
  find_txt,
  find_case,
  iup.hbox{
    iup.fill{},
    bt_find_next,
    bt_find_close,
    normalizesize="HORIZONTAL",
  },
  margin = "10x10",
  gap = "5",
```

```
    }

    find_dlg = iup.dialog{
      box,
      title = "Find",
      dialogframe = "Yes",
      defaultenter = bt_next,
      defaultesc = bt_close,
      parentdialog = iup.GetDialog(self)
      }

    -- Save the dialog to reuse it
    self.find_dialog = find_dlg -- from the main dialog */
  end

  -- centerparent first time, next time reuse the last position
  find_dlg:showxy(iup.CURRENT, iup.CURRENT)
end

function item_copy:action()
  local clipboard = iup.clipboard{text = multitext.selectedtext}
  clipboard:destroy()
end

function item_paste:action()
  local clipboard = iup.clipboard{}
  multitext.insert = clipboard.text
  clipboard:destroy()
  return iup.IGNORE -- avoid system processing for hot keys, to correctly parse li
end

function item_cut:action()
  local clipboard = iup.clipboard{text = multitext.selectedtext}
  multitext.selectedtext = ""
  clipboard:destroy()
end

function item_delete:action()
  multitext.selectedtext = ""
end

function item_select_all:action()
```

```
  iup.SetFocus(multitext)
  multitext.selection = "ALL"
end

function item_font:action()
  local font = multitext.font
  local fontdlg = iup.fontdlg{value = font, parentdialog=iup.GetDialog(self)}

  fontdlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(fontdlg.status) == 1) then
    multitext.font = fontdlg.value
    config:SetVariable("MainWindow", "Font", fontdlg.value)
  end

  fontdlg:destroy()
end

function item_about:action()
  iup.Message("About", "   Simple Notepad\n\nAutors:\n   Gustavo Lyrio\n   Antonio 
end


--********************************* Main (Part 2/2) ****************************


recent_menu = iup.menu{}

file_menu = iup.menu{
  item_open,
  item_saveas,
  iup.separator{},
  iup.submenu{title="Recent &Files", recent_menu},
  item_exit
  }

edit_menu = iup.menu{
    item_cut,
    item_copy,
    item_paste,
    item_delete,
    iup.separator{},
```

```
    item_find,
    item_goto,
    iup.separator{},
    item_select_all
    }

function edit_menu:open_cb()
  local clipboard = iup.clipboard{}

  if (not clipboard.textavailable) then
    item_paste.active = "NO"
  else
    item_paste.active = "YES"
  end

  if (not multitext.selectedtext) then
    item_cut.active = "NO"
    item_delete.active = "NO"
    item_copy.active = "NO"
  else
    item_cut.active = "YES"
    item_delete.active = "YES"
    item_copy.active = "YES"
  end

  clipboard:destroy()
end

format_menu = iup.menu{item_font}
help_menu = iup.menu{item_about}
sub_menu_file = iup.submenu{file_menu, title = "&File"}
sub_menu_edit = iup.submenu{edit_menu, title = "&Edit"}
sub_menu_format = iup.submenu{format_menu, title = "F&ormat"}
sub_menu_help = iup.submenu{help_menu, title = "&Help"}

menu = iup.menu{
  sub_menu_file,
  sub_menu_edit,
  sub_menu_format,
  sub_menu_help,
  }
```

```
btn_open = iup.button{image = "IUP_FileOpen", flat = "Yes", action = item_open.actio
btn_save = iup.button{image = "IUP_FileSave", flat = "Yes", action = item_saveas.ac
btn_find = iup.button{image = "IUP_EditFind", flat = "Yes", action = item_find.actio

toolbar_hb = iup.hbox{
  btn_open,
  btn_save,
  iup.label{separator="VERTICAL"},
  btn_find,
  margin = "5x5",
  gap = 2,
}

vbox = iup.vbox{
  toolbar_hb,
  multitext,
  lbl_statusbar,
}

dlg = iup.dialog{
  vbox,
  title = "Simple Notepad",
  size = "HALFxHALF",
  menu = menu,
  close_cb = item_exit.action,
}

function dlg:k_any(c)
  if (c == iup.K_cO) then
    item_open:action()
  elseif (c == iup.K_cS) then
    item_saveas:action()
  elseif (c == iup.K_cF) then
    item_find:action()
  elseif (c == iup.K_cG) then
    item_goto:action()
  end
end

config:RecentInit(recent_menu, 10)

-- parent for pre-defined dialogs in closed functions (IupMessage)
```

```
iup.SetGlobal("PARENTDIALOG", iup.SetHandleName(dlg))

config:DialogShow(dlg, "MainWindow")

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

3.9 文件管理

在本节，我们将看到更多有关文件的处理。本例会显示如何来处理拖拽，如何处理程序的命令行，以及某个操作执行之前文件是否需要保存。

首先，我们解释 *str_filetilte*，*new_file*，*open_file*,*save_file* 和 *save_check* 函数。

str_filetitle 函数被用来添加文件名到程序的对话框窗口。*new_file* 首先获取主窗口和文本框的句柄，然后设置窗口标题为 *Untitled - Simple Notepad*。设置文本框的 *FILENAME* 属性为 *NULL*，*VALUE* 属性为""，最后设置一个新的属性 *DIRTY* 的值为 *NO*。*DIRTY* 是我们创建的一个自定义的属性，它被我们用来检查文本框内容是否有变化，从而使我们能够提示用户是否需要保存。*open_file* 函数用来读取文件，它设置的属性和 *new_file* 函数几乎是相同的，除了它使用 *str_filetitle* 来设置程序的窗口标题，并把文本框的 *VALUE* 值设置为文件内容。需要注意的是，和 *new_file* 一样，它也要设置 *DIRTY* 属性值为 *NO*，毕竟读取文件之后，我们并没有修改它，所以就不要提示保存文件。*open_file* 调用 *IupConfigRecentUpdate* 函数来包含我们刚刚打开的文件到最近文件列表中。*save_file* 调用 *write_file* 函数把文本框的内容写入文件中，然后设置 *DIRTY* 属性为 *NO*。*saveas_file* 函数进行同样的操作，只是会弹出一个对话框来让用户选择要写入的文件地址，然后更新最近文件列表。最后，*save_check* 使用 *DIRTY* 属性来检查文本框的内容是否需要保存。我们在这里使用 *IupGetInt* 函数来自动转换 *DIRTY* 属性的 *YES*，*NO* 到 1 或 0。如果是 1，我们调用一个预定义对话框 *IupAlarm* 来警告用户，文本框的内容已经变化了，需要保存。如果用户选择 *YES* 按钮，就会调用 *item_save_action_cb* 来保存文件。如果用户选择 *NO*，*save_check* 函数会返回 1，然后不进行保存操作，继续执行用户的操作，如果用户选择 *Cancel* 按钮，*save_check* 函数会返回 0，之后不会进行任何操作。

接下来，我们看回调部分。我们添加了几个新的回调。首先是 *dropfiles_cb*，它对拖拽到程序中的文件进行处理。这个回调十分简单，它先使用 *save_check* 函数检查当前文本框的内容是否需要保存，然后调用 *open_file* 函数打开文件。需要注意的是这个回调同时被主窗口和文本框的 *DROPFILE_CB* 绑定。这样做是因为，拖拽文件到窗口上的控件，除了文本框之外的控件都会调用窗口的 *DROPFILE_CB*，所以需要为文本框特别指定这个回调。

接下来是 *multitext_valuechanged_cb* 回调函数，它在文本框的内容被改变之后被调用。在这里，我们把 *DIRTY* 属性设置为 *YES*。这个回调函数和文本框的 *VALUECHANGED_CB* 回调相关联。

然后是 *file_menu_open_cb* 回调函数，它在用户点击文件菜单，菜单还没有显示之前被调用。我们使用它结合 *DIRTY* 属性来禁用，解禁保存恢复菜单项目。

我们修改了 *config_recent_cb* 回调函数，现在它会检查当前文本框的内容是否需要保存，并且它调用了我们添加的 *open_file* 函数。另一个新的回调函数是 *item_new_action_cb*，它做了相同的

工作，只是调用 *new_file* 而不是 *open_file* 函数。*item_open_action_cb* 和 *item_exit_action_cb* 回调函数都被修改使用 *save_check* 和 *open_file* 函数。

我们还创建了另外两个回调函数：*item_save_action_cb* 和 *item_revert_action_cb* 回调函数。*item_save_action_cb* 回调函数使用保存在 *FILENAME* 属性的当前文件名来存储文本框的内容它不会弹出另存为对话框。如果不存在当前文件名，就调用 *item_saveas_action_cb* 函数来让用户选择一个文件名进行保存。*item_revert_action_cb* 函数会忽略掉文本框内容的修改，重新载入文件。

在主函数，我们添加了几个新的菜单项目:*item_new*，*item_save* 和 *item_revert*。我们也添加了几个新的按钮:*btn_cnt*，**btn_copy* ，*i*btn_paste* 和 *btn_new*。文本框现在有新的 *DIRTY* 属性和两个新的回调:*VALUECHANGED_CB* 和 *DROPFILES_CB*。我们调用 *new_file* 来初始化程序，就像一个新文件刚刚被创建一样。这里，我们还重新组织了代码，使代码看上去结构更清晰。最后，我们使用 *argv* 和 *argv* 来检查命令行，如果用户使用命令行指定了一个文件，我们就调用 *open_file* 打开它。

到目前，文件相关处理已经完结，下一节，我们进行控件动态布局的讨论。

C语言

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <iup.h>
#include <iup_config.h>


/********************************* Utilities *************************************


const char* str_filetitle(const char *filename)
{
  /* Start at the last character */
  int len = (int)strlen(filename);
  int offset = len - 1;
  while (offset != 0)
  {
    if (filename[offset] == '\\' || filename[offset] == '/')
    {
      offset++;
      break;
    }
    offset--;
  }
  return filename + offset;
}
```

```
int str_compare(const char *l, const char *r, int casesensitive)
{
  if (!l || !r)
    return 0;

  while (*l && *r)
  {
    int diff;
    char l_char = *l,
      r_char = *r;

    /* compute the difference of both characters */
    if (casesensitive)
      diff = l_char - r_char;
    else
      diff = tolower((int)l_char) - tolower((int)r_char);

    /* if they differ we have a result */
    if (diff != 0)
      return 0;

    /* otherwise process the next characters */
    ++l;
    ++r;
  }

  /* check also for terminator */
  if (*l == *r)
    return 1;

  if (*r == 0)
    return 1;  /* if second string is at terminator, then it is partially equal */

  return 0;
}

int str_find(const char *str, const char *str_to_find, int casesensitive)
{
  int i, str_len, str_to_find_len, count;

  if (!str || str[0] == 0 || !str_to_find || str_to_find[0] == 0)
    return -1;
```

```
  str_len = (int)strlen(str);
  str_to_find_len = (int)strlen(str_to_find);
  count = str_len - str_to_find_len;
  if (count < 0)
    return -1;

  count++;

  for (i = 0; i<count; i++)
  {
    if (str_compare(str, str_to_find, casesensitive))
      return i;

    str++;
  }

  return -1;
}

char* read_file(const char* filename)
{
  int size;
  char* str;
  FILE* file = fopen(filename, "rb");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return NULL;
  }

  /* calculate file size */
  fseek(file, 0, SEEK_END);
  size = ftell(file);
  fseek(file, 0, SEEK_SET);

  /* allocate memory for the file contents + nul terminator */
  str = malloc(size + 1);
  /* read all data at once */
  fread(str, size, 1, file);
  /* set the nul terminator */
  str[size] = 0;
```

```
  if (ferror(file))
    IupMessagef("Error", "Fail when reading from file: %s", filename);

  fclose(file);
  return str;
}

int write_file(const char* filename, const char* str, int count)
{
  FILE* file = fopen(filename, "w");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return 0;
  }

  fwrite(str, 1, count, file);

  if (ferror(file))
    IupMessagef("Error", "Fail when writing to file: %s", filename);

  fclose(file);
  return 1;
}

void new_file(Ihandle* ih)
{
  Ihandle* dlg = IupGetDialog(ih);
  Ihandle* multitext = IupGetDialogChild(dlg, "MULTITEXT");

  IupSetAttribute(dlg, "TITLE", "Untitled - Simple Notepad");
  IupSetAttribute(multitext, "FILENAME", NULL);
  IupSetAttribute(multitext, "DIRTY", "NO");
  IupSetAttribute(multitext, "VALUE", "");
}

void open_file(Ihandle* ih, const char* filename)
{
  char* str = read_file(filename);
  if (str)
  {
```

```
    Ihandle* dlg = IupGetDialog(ih);
    Ihandle* multitext = IupGetDialogChild(dlg, "MULTITEXT");
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

    IupSetfAttribute(dlg, "TITLE", "%s - Simple Notepad", str_filetitle(filename));
    IupSetStrAttribute(multitext, "FILENAME", filename);
    IupSetAttribute(multitext, "DIRTY", "NO");
    IupSetStrAttribute(multitext, "VALUE", str);

    IupConfigRecentUpdate(config, filename);

    free(str);
  }
}

void save_file(Ihandle* multitext)
{
  char* filename = IupGetAttribute(multitext, "FILENAME");
  char* str = IupGetAttribute(multitext, "VALUE");
  int count = IupGetInt(multitext, "COUNT");
  if (write_file(filename, str, count))
    IupSetAttribute(multitext, "DIRTY", "NO");
}

void saveas_file(Ihandle* multitext, const char* filename)
{
  char* str = IupGetAttribute(multitext, "VALUE");
  int count = IupGetInt(multitext, "COUNT");
  if (write_file(filename, str, count))
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

    IupSetfAttribute(IupGetDialog(multitext), "TITLE", "%s - Simple Notepad", str_
    IupSetStrAttribute(multitext, "FILENAME", filename);
    IupSetAttribute(multitext, "DIRTY", "NO");

    IupConfigRecentUpdate(config, filename);
  }
}

int save_check(Ihandle* ih)
{
```

```
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");
  if (IupGetInt(multitext, "DIRTY"))
  {
    switch (IupAlarm("Warning", "File not saved! Save it now?", "Yes", "No", "Cance
    {
    case 1:  /* save the changes and continue */
      save_file(multitext);
      break;
    case 2:  /* ignore the changes and continue */
      break;
    case 3:  /* cancel */
      return 0;
    }
  }
  return 1;
}


/********************************* Callbacks *************************************


int dropfiles_cb(Ihandle* ih, const char* filename)
{
  if (save_check(ih))
    open_file(ih, filename);

  return IUP_DEFAULT;
}

int multitext_valuechanged_cb(Ihandle* multitext)
{
  IupSetAttribute(multitext, "DIRTY", "YES");
  return IUP_DEFAULT;
}

int file_menu_open_cb(Ihandle* ih)
{
  Ihandle* item_revert = IupGetDialogChild(ih, "ITEM_REVERT");
  Ihandle* item_save = IupGetDialogChild(ih, "ITEM_SAVE");
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");
  char* filename = IupGetAttribute(multitext, "FILENAME");
  int dirty = IupGetInt(multitext, "DIRTY");
```

```
  if (dirty)
    IupSetAttribute(item_save, "ACTIVE", "YES");
  else
    IupSetAttribute(item_save, "ACTIVE", "NO");

  if (dirty && filename)
    IupSetAttribute(item_revert, "ACTIVE", "YES");
  else
    IupSetAttribute(item_revert, "ACTIVE", "NO");
  return IUP_DEFAULT;
}

int edit_menu_open_cb(Ihandle* ih)
{
  Ihandle *clipboard = IupClipboard();

  Ihandle *item_paste = IupGetDialogChild(ih, "ITEM_PASTE");
  Ihandle *item_cut = IupGetDialogChild(ih, "ITEM_CUT");
  Ihandle *item_delete = IupGetDialogChild(ih, "ITEM_DELETE");
  Ihandle *item_copy = IupGetDialogChild(ih, "ITEM_COPY");
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  if (!IupGetInt(clipboard, "TEXTAVAILABLE"))
    IupSetAttribute(item_paste, "ACTIVE", "NO");
  else
    IupSetAttribute(item_paste, "ACTIVE", "YES");

  if (!IupGetAttribute(multitext, "SELECTEDTEXT"))
  {
    IupSetAttribute(item_cut, "ACTIVE", "NO");
    IupSetAttribute(item_delete, "ACTIVE", "NO");
    IupSetAttribute(item_copy, "ACTIVE", "NO");
  }
  else
  {
    IupSetAttribute(item_cut, "ACTIVE", "YES");
    IupSetAttribute(item_delete, "ACTIVE", "YES");
    IupSetAttribute(item_copy, "ACTIVE", "YES");
  }

  IupDestroy(clipboard);
```

```
  return IUP_DEFAULT;
}

int config_recent_cb(Ihandle* ih)
{
  if (save_check(ih))
  {
    char* filename = IupGetAttribute(ih, "TITLE");
    open_file(ih, filename);
  }
  return IUP_DEFAULT;
}

int multitext_caret_cb(Ihandle *ih, int lin, int col)
{
  Ihandle *lbl_statusbar = IupGetDialogChild(ih, "STATUSBAR");
  IupSetfAttribute(lbl_statusbar, "TITLE", "Lin %d, Col %d", lin, col);
  return IUP_DEFAULT;
}

int item_new_action_cb(Ihandle* item_new)
{
  if (save_check(item_new))
    new_file(item_new);

  return IUP_DEFAULT;
}

int item_open_action_cb(Ihandle* item_open)
{
  Ihandle *filedlg;

  if (!save_check(item_open))
    return IUP_DEFAULT;

  filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "OPEN");
  IupSetAttribute(filedlg, "EXTFILTER", "Text Files|*.txt|All Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_open));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
  if (IupGetInt(filedlg, "STATUS") != -1)
```

```
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    open_file(item_open, filename);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_saveas_action_cb(Ihandle* item_saveas)
{
  Ihandle* multitext = IupGetDialogChild(item_saveas, "MULTITEXT");
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "SAVE");
  IupSetAttribute(filedlg, "EXTFILTER", "Text␣Files|*.txt|All␣Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_saveas));
  IupSetStrAttribute(filedlg, "FILE", IupGetAttribute(multitext, "FILENAME"));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    saveas_file(multitext, filename);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_save_action_cb(Ihandle* item_save)
{
  Ihandle* multitext = IupGetDialogChild(item_save, "MULTITEXT");
  char* filename = IupGetAttribute(multitext, "FILENAME");
  if (!filename)
    item_saveas_action_cb(item_save);
  else
    save_file(multitext);
  return IUP_DEFAULT;
}

int item_revert_action_cb(Ihandle* item_revert)
```

```
{
  Ihandle* multitext = IupGetDialogChild(item_revert, "MULTITEXT");
  char* filename = IupGetAttribute(multitext, "FILENAME");
  open_file(item_revert, filename);
  return IUP_DEFAULT;
}

int item_exit_action_cb(Ihandle* item_exit)
{
  Ihandle* dlg = IupGetDialog(item_exit);
  Ihandle* config = (Ihandle*)IupGetAttribute(dlg, "CONFIG");

  if (!save_check(item_exit))
    return IUP_IGNORE;  /* to abort the CLOSE_CB callback */

  IupConfigDialogClosed(config, dlg, "MainWindow");
  IupConfigSave(config);
  IupDestroy(config);
  return IUP_CLOSE;
}

int goto_ok_action_cb(Ihandle* bt_ok)
{
  int line_count = IupGetInt(bt_ok, "TEXT_LINECOUNT");
  Ihandle* txt = IupGetDialogChild(bt_ok, "LINE_TEXT");
  int line = IupGetInt(txt, "VALUE");
  if (line < 1 || line >= line_count)
  {
    IupMessage("Error", "Invalid line number.");
    return IUP_DEFAULT;
  }

  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "1");
  return IUP_CLOSE;
}

int goto_cancel_action_cb(Ihandle* bt_ok)
{
  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "0");
  return IUP_CLOSE;
}
```

```
int item_goto_action_cb(Ihandle* item_goto)
{
  Ihandle* multitext = IupGetDialogChild(item_goto, "MULTITEXT");
  Ihandle *dlg, *box, *bt_ok, *bt_cancel, *txt, *lbl;
  int line_count = IupGetInt(multitext, "LINECOUNT");

  lbl = IupLabel(NULL);
  IupSetfAttribute(lbl, "TITLE", "Line Number [1-%d]:", line_count);
  txt = IupText(NULL);
  IupSetAttribute(txt, "MASK", IUP_MASK_UINT);  /* unsigned integer numbers only */
  IupSetAttribute(txt, "NAME", "LINE_TEXT");
  IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
  bt_ok = IupButton("OK", NULL);
  IupSetInt(bt_ok, "TEXT_LINECOUNT", line_count);
  IupSetAttribute(bt_ok, "PADDING", "10x2");
  IupSetCallback(bt_ok, "ACTION", (Icallback)goto_ok_action_cb);
  bt_cancel = IupButton("Cancel", NULL);
  IupSetCallback(bt_cancel, "ACTION", (Icallback)goto_cancel_action_cb);
  IupSetAttribute(bt_cancel, "PADDING", "10x2");

  box = IupVbox(
    lbl,
    txt,
    IupSetAttributes(IupHbox(
      IupFill(),
      bt_ok,
      bt_cancel,
      NULL), "NORMALIZESIZE=HORIZONTAL"),
    NULL);
  IupSetAttribute(box, "MARGIN", "10x10");
  IupSetAttribute(box, "GAP", "5");

  dlg = IupDialog(box);
  IupSetAttribute(dlg, "TITLE", "Go To Line");
  IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
  IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_ok);
  IupSetAttributeHandle(dlg, "DEFAULTESC", bt_cancel);
  IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_goto));

  IupPopup(dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(dlg, "STATUS") == 1)
```

```
  {
    int line = IupGetInt(txt, "VALUE");
    int pos;
    IupTextConvertLinColToPos(multitext, line, 0, &pos);
    IupSetInt(multitext, "CARETPOS", pos);
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }

  IupDestroy(dlg);

  return IUP_DEFAULT;
}

int find_next_action_cb(Ihandle* bt_next)
{
  Ihandle* multitext = (Ihandle*)IupGetAttribute(bt_next, "MULTITEXT");
  char* str = IupGetAttribute(multitext, "VALUE");
  int find_pos = IupGetInt(multitext, "FIND_POS");

  Ihandle* txt = IupGetDialogChild(bt_next, "FIND_TEXT");
  char* str_to_find = IupGetAttribute(txt, "VALUE");

  Ihandle* find_case = IupGetDialogChild(bt_next, "FIND_CASE");
  int casesensitive = IupGetInt(find_case, "VALUE");

  int pos = str_find(str + find_pos, str_to_find, casesensitive);
  if (pos >= 0)
    pos += find_pos;
  else if (find_pos > 0)
    pos = str_find(str, str_to_find, casesensitive);  /* try again from the start *

  if (pos >= 0)
  {
    int lin, col,
      end_pos = pos + (int)strlen(str_to_find);

    IupSetInt(multitext, "FIND_POS", end_pos);

    IupSetFocus(multitext);
    IupSetfAttribute(multitext, "SELECTIONPOS", "%d:%d", pos, end_pos);

    IupTextConvertPosToLinCol(multitext, pos, &lin, &col);
```

```
    IupTextConvertLinColToPos(multitext, lin, 0, &pos);  /* position at col=0, just
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }
  else
    IupMessage("Warning", "Text not found.");

  return IUP_DEFAULT;
}

int find_close_action_cb(Ihandle* bt_close)
{
  IupHide(IupGetDialog(bt_close));
  return IUP_DEFAULT;
}

int item_find_action_cb(Ihandle* item_find)
{
  Ihandle* dlg = (Ihandle*)IupGetAttribute(item_find, "FIND_DIALOG");
  if (!dlg)
  {
    Ihandle* multitext = IupGetDialogChild(item_find, "MULTITEXT");
    Ihandle *box, *bt_next, *bt_close, *txt, *find_case;

    txt = IupText(NULL);
    IupSetAttribute(txt, "NAME", "FIND_TEXT");
    IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
    find_case = IupToggle("Case Sensitive", NULL);
    IupSetAttribute(find_case, "NAME", "FIND_CASE");
    bt_next = IupButton("Find Next", NULL);
    IupSetAttribute(bt_next, "PADDING", "10x2");
    IupSetCallback(bt_next, "ACTION", (Icallback)find_next_action_cb);
    bt_close = IupButton("Close", NULL);
    IupSetCallback(bt_close, "ACTION", (Icallback)find_close_action_cb);
    IupSetAttribute(bt_close, "PADDING", "10x2");

    box = IupVbox(
      IupLabel("Find What:"),
      txt,
      find_case,
      IupSetAttributes(IupHbox(
        IupFill(),
        bt_next,
```

```
        bt_close,
        NULL), "NORMALIZESIZE=HORIZONTAL"),
      NULL);
    IupSetAttribute(box, "MARGIN", "10x10");
    IupSetAttribute(box, "GAP", "5");

    dlg = IupDialog(box);
    IupSetAttribute(dlg, "TITLE", "Find");
    IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
    IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_next);
    IupSetAttributeHandle(dlg, "DEFAULTESC", bt_close);
    IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_find));

    /* Save the multiline to acess it from the callbacks */
    IupSetAttribute(dlg, "MULTITEXT", (char*)multitext);

    /* Save the dialog to reuse it */
    IupSetAttribute(item_find, "FIND_DIALOG", (char*)dlg);
  }

  /* centerparent first time, next time reuse the last position */
  IupShowXY(dlg, IUP_CURRENT, IUP_CURRENT);

  return IUP_DEFAULT;
}

int item_copy_action_cb(Ihandle* item_copy)
{
  Ihandle* multitext = IupGetDialogChild(item_copy, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(clipboard, "TEXT", IupGetAttribute(multitext, "SELECTEDTEXT"));
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_paste_action_cb(Ihandle* item_paste)
{
  Ihandle* multitext = IupGetDialogChild(item_paste, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(multitext, "INSERT", IupGetAttribute(clipboard, "TEXT"));
  IupDestroy(clipboard);
  return IUP_DEFAULT;
```

```
}

int item_cut_action_cb(Ihandle* item_cut)
{
  Ihandle* multitext = IupGetDialogChild(item_cut, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(clipboard, "TEXT", IupGetAttribute(multitext, "SELECTEDTEXT"));
  IupSetAttribute(multitext, "SELECTEDTEXT", "");
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_delete_action_cb(Ihandle* item_delete)
{
  Ihandle* multitext = IupGetDialogChild(item_delete, "MULTITEXT");
  IupSetAttribute(multitext, "SELECTEDTEXT", "");
  return IUP_DEFAULT;
}

int item_select_all_action_cb(Ihandle* item_select_all)
{
  Ihandle* multitext = IupGetDialogChild(item_select_all, "MULTITEXT");
  IupSetFocus(multitext);
  IupSetAttribute(multitext, "SELECTION", "ALL");
  return IUP_DEFAULT;
}

int item_font_action_cb(Ihandle* item_font)
{
  Ihandle* multitext = IupGetDialogChild(item_font, "MULTITEXT");
  Ihandle* fontdlg = IupFontDlg();
  char* font = IupGetAttribute(multitext, "FONT");
  IupSetStrAttribute(fontdlg, "VALUE", font);
  IupSetAttributeHandle(fontdlg, "PARENTDIALOG", IupGetDialog(item_font));

  IupPopup(fontdlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(fontdlg, "STATUS") == 1)
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
    font = IupGetAttribute(fontdlg, "VALUE");
    IupSetStrAttribute(multitext, "FONT", font);
```

```
    IupConfigSetVariableStr(config, "MainWindow", "Font", font);
  }

  IupDestroy(fontdlg);
  return IUP_DEFAULT;
}

int item_about_action_cb(void)
{
  IupMessage("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣S
  return IUP_DEFAULT;
}


/********************************** Main ****************************************/


int main(int argc, char **argv)
{
  Ihandle *dlg, *vbox, *multitext, *menu;
  Ihandle *sub_menu_file, *file_menu, *item_exit, *item_new, *item_open, *item_save
  Ihandle *sub_menu_edit, *edit_menu, *item_find, *item_goto, *item_copy, *item_pas
  Ihandle *btn_cut, *btn_copy, *btn_paste, *btn_find, *btn_new, *btn_open, *btn_sav
  Ihandle *sub_menu_format, *format_menu, *item_font;
  Ihandle *sub_menu_help, *help_menu, *item_about;
  Ihandle *lbl_statusbar, *toolbar_hb, *recent_menu;
  Ihandle *config;
  const char* font;

  IupOpen(&argc, &argv);
  IupImageLibOpen();

  config = IupConfig();
  IupSetAttribute(config, "APP_NAME", "simple_notepad");
  IupConfigLoad(config);

  multitext = IupText(NULL);
    IupSetAttribute(multitext, "MULTILINE", "YES");
    IupSetAttribute(multitext, "EXPAND", "YES");
    IupSetAttribute(multitext, "NAME", "MULTITEXT");
    IupSetAttribute(multitext, "DIRTY", "NO");
```

```
  IupSetCallback(multitext, "CARET_CB", (Icallback)multitext_caret_cb);
  IupSetCallback(multitext, "VALUECHANGED_CB", (Icallback)multitext_valuechanged_
  IupSetCallback(multitext, "DROPFILES_CB", (Icallback)dropfiles_cb);

font = IupConfigGetVariableStr(config, "MainWindow", "Font");
if (font)
  IupSetStrAttribute(multitext, "FONT", font);

lbl_statusbar = IupLabel("Lin 1, Col 1");
  IupSetAttribute(lbl_statusbar, "NAME", "STATUSBAR");
  IupSetAttribute(lbl_statusbar, "EXPAND", "HORIZONTAL");
  IupSetAttribute(lbl_statusbar, "PADDING", "10x5");

item_new = IupItem("New\tCtrl+N", NULL);
  IupSetAttribute(item_new, "IMAGE", "IUP_FileNew");
  IupSetCallback(item_new, "ACTION", (Icallback)item_new_action_cb);
btn_new = IupButton(NULL, NULL);
  IupSetAttribute(btn_new, "IMAGE", "IUP_FileNew");
  IupSetAttribute(btn_new, "FLAT", "Yes");
  IupSetCallback(btn_new, "ACTION", (Icallback)item_new_action_cb);
  IupSetAttribute(btn_new, "TIP", "New (Ctrl+N)");
  IupSetAttribute(btn_new, "CANFOCUS", "No");

item_open = IupItem("&Open...\tCtrl+O", NULL);
  IupSetAttribute(item_open, "IMAGE", "IUP_FileOpen");
  IupSetCallback(item_open, "ACTION", (Icallback)item_open_action_cb);
btn_open = IupButton(NULL, NULL);
  IupSetAttribute(btn_open, "IMAGE", "IUP_FileOpen");
  IupSetAttribute(btn_open, "FLAT", "Yes");
  IupSetCallback(btn_open, "ACTION", (Icallback)item_open_action_cb);
  IupSetAttribute(btn_open, "TIP", "Open (Ctrl+O)");
  IupSetAttribute(btn_open, "CANFOCUS", "No");

item_save = IupItem("Save\tCtrl+S", NULL);
  IupSetAttribute(item_save, "NAME", "ITEM_SAVE");
  IupSetAttribute(item_save, "IMAGE", "IUP_FileSave");
  IupSetCallback(item_save, "ACTION", (Icallback)item_save_action_cb);
btn_save = IupButton(NULL, NULL);
  IupSetAttribute(btn_save, "IMAGE", "IUP_FileSave");
  IupSetAttribute(btn_save, "FLAT", "Yes");
  IupSetCallback(btn_save, "ACTION", (Icallback)item_save_action_cb);
  IupSetAttribute(btn_save, "TIP", "Save (Ctrl+S)");
```

```
  IupSetAttribute(btn_save, "CANFOCUS", "No");

item_saveas = IupItem("Save␣&As...", NULL);
  IupSetAttribute(item_saveas, "NAME", "ITEM_SAVEAS");
  IupSetCallback(item_saveas, "ACTION", (Icallback)item_saveas_action_cb);

item_revert = IupItem("Revert", NULL);
  IupSetAttribute(item_revert, "NAME", "ITEM_REVERT");
  IupSetCallback(item_revert, "ACTION", (Icallback)item_revert_action_cb);

item_exit = IupItem("E&xit", NULL);
  IupSetCallback(item_exit, "ACTION", (Icallback)item_exit_action_cb);

item_find = IupItem("&Find...\tCtrl+F", NULL);
  IupSetAttribute(item_find, "IMAGE", "IUP_EditFind");
  IupSetCallback(item_find, "ACTION", (Icallback)item_find_action_cb);
btn_find = IupButton(NULL, NULL);
  IupSetAttribute(btn_find, "IMAGE", "IUP_EditFind");
  IupSetAttribute(btn_find, "FLAT", "Yes");
  IupSetCallback(btn_find, "ACTION", (Icallback)item_find_action_cb);
  IupSetAttribute(btn_find, "TIP", "Find␣(Ctrl+F)");
  IupSetAttribute(btn_find, "CANFOCUS", "No");

item_cut = IupItem("Cut\tCtrl+X", NULL);
  IupSetAttribute(item_cut, "NAME", "ITEM_CUT");
  IupSetAttribute(item_cut, "IMAGE", "IUP_EditCut");
  IupSetCallback(item_cut, "ACTION", (Icallback)item_cut_action_cb);
item_copy = IupItem("Copy\tCtrl+C", NULL);
  IupSetAttribute(item_copy, "NAME", "ITEM_COPY");
  IupSetAttribute(item_copy, "IMAGE", "IUP_EditCopy");
  IupSetCallback(item_copy, "ACTION", (Icallback)item_copy_action_cb);
item_paste = IupItem("Paste\tCtrl+V", NULL);
  IupSetAttribute(item_paste, "NAME", "ITEM_PASTE");
  IupSetAttribute(item_paste, "IMAGE", "IUP_EditPaste");
  IupSetCallback(item_paste, "ACTION", (Icallback)item_paste_action_cb);
item_delete = IupItem("Delete\tDel", NULL);
  IupSetAttribute(item_delete, "IMAGE", "IUP_EditErase");
  IupSetAttribute(item_delete, "NAME", "ITEM_DELETE");
  IupSetCallback(item_delete, "ACTION", (Icallback)item_delete_action_cb);
item_select_all = IupItem("Select␣All\tCtrl+A", NULL);
  IupSetCallback(item_select_all, "ACTION", (Icallback)item_select_all_action_cb)
```

```
btn_cut = IupButton(NULL, NULL);
  IupSetAttribute(btn_cut, "IMAGE", "IUP_EditCut");
  IupSetAttribute(btn_cut, "FLAT", "Yes");
  IupSetCallback(btn_cut, "ACTION", (Icallback)item_cut_action_cb);
  IupSetAttribute(btn_cut, "TIP", "Cut (Ctrl+X)");
  IupSetAttribute(btn_cut, "CANFOCUS", "No");
btn_copy = IupButton(NULL, NULL);
  IupSetAttribute(btn_copy, "IMAGE", "IUP_EditCopy");
  IupSetAttribute(btn_copy, "FLAT", "Yes");
  IupSetCallback(btn_copy, "ACTION", (Icallback)item_copy_action_cb);
  IupSetAttribute(btn_copy, "TIP", "Copy (Ctrl+C)");
  IupSetAttribute(btn_copy, "CANFOCUS", "No");
btn_paste = IupButton(NULL, NULL);
  IupSetAttribute(btn_paste, "IMAGE", "IUP_EditPaste");
  IupSetAttribute(btn_paste, "FLAT", "Yes");
  IupSetCallback(btn_paste, "ACTION", (Icallback)item_paste_action_cb);
  IupSetAttribute(btn_paste, "TIP", "Paste (Ctrl+V)");
  IupSetAttribute(btn_paste, "CANFOCUS", "No");

toolbar_hb = IupHbox(
  btn_new,
  btn_open,
  btn_save,
  IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
  btn_cut,
  btn_copy,
  btn_paste,
  IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
  btn_find,
  NULL);
IupSetAttribute(toolbar_hb, "MARGIN", "5x5");
IupSetAttribute(toolbar_hb, "GAP", "2");

item_goto = IupItem("&Go To...\tCtrl+G", NULL);
  IupSetCallback(item_goto, "ACTION", (Icallback)item_goto_action_cb);
item_font = IupItem("&Font...", NULL);
  IupSetCallback(item_font, "ACTION", (Icallback)item_font_action_cb);
item_about = IupItem("&About...", NULL);
  IupSetCallback(item_about, "ACTION", (Icallback)item_about_action_cb);

recent_menu = IupMenu(NULL);
```

```
file_menu = IupMenu(
  item_new,
  item_open,
  item_save,
  item_saveas,
  item_revert,
  IupSeparator(),
  IupSubmenu("Recent &Files", recent_menu),
  item_exit,
  NULL);
edit_menu = IupMenu(
  item_cut,
  item_copy,
  item_paste,
  item_delete,
  IupSeparator(),
  item_find,
  item_goto,
  IupSeparator(),
  item_select_all,
  NULL);
format_menu = IupMenu(
  item_font,
  NULL);
help_menu = IupMenu(
  item_about,
  NULL);

IupSetCallback(file_menu, "OPEN_CB", (Icallback)file_menu_open_cb);
IupSetCallback(edit_menu, "OPEN_CB", (Icallback)edit_menu_open_cb);

sub_menu_file = IupSubmenu("&File", file_menu);
sub_menu_edit = IupSubmenu("&Edit", edit_menu);
sub_menu_format = IupSubmenu("F&ormat", format_menu);
sub_menu_help = IupSubmenu("&Help", help_menu);

menu = IupMenu(
  sub_menu_file,
  sub_menu_edit,
  sub_menu_format,
  sub_menu_help,
  NULL);
```

```
  vbox = IupVbox(
    toolbar_hb,
    multitext,
    lbl_statusbar,
    NULL);

  dlg = IupDialog(vbox);
  IupSetAttributeHandle(dlg, "MENU", menu);
  IupSetAttribute(dlg, "SIZE", "HALFxHALF");
  IupSetCallback(dlg, "CLOSE_CB", (Icallback)item_exit_action_cb);
  IupSetCallback(dlg, "DROPFILES_CB", (Icallback)dropfiles_cb);

  IupSetAttribute(dlg, "CONFIG", (char*)config);

  /* parent for pre-defined dialogs in closed functions (IupMessage) */
  IupSetAttributeHandle(NULL, "PARENTDIALOG", dlg);

  IupSetCallback(dlg, "K_cN", (Icallback)item_new_action_cb);
  IupSetCallback(dlg, "K_cO", (Icallback)item_open_action_cb);
  IupSetCallback(dlg, "K_cS", (Icallback)item_save_action_cb);
  IupSetCallback(dlg, "K_cF", (Icallback)item_find_action_cb);
  IupSetCallback(dlg, "K_cG", (Icallback)item_goto_action_cb);

  IupConfigRecentInit(config, recent_menu, config_recent_cb, 10);

  IupConfigDialogShow(config, dlg, "MainWindow");

  /* initialize the current file */
  new_file(dlg);

  /* open a file from the command line (allow file association in Windows) */
  if (argc > 1 && argv[1])
  {
    const char* filename = argv[1];
    open_file(dlg, filename);
  }

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
```

}
Lua语言

```
require("iuplua")
require("iupluaimglib")


--********************************* Utilities *********************************


function str_find(str, str_to_find, casesensitive, start)
  if (not casesensitive) then
    return str_find(string.lower(str), string.lower(str_to_find), true, start)
  end

  return string.find(str, str_to_find, start, true)
end

function str_filetitle(filename)
  local filename = string.gsub(filename, "\\", "/")
  filename = string.reverse(filename)
  final = string.find(filename, '/')
  filename = string.sub(filename, 1, final-1)
  filename = string.reverse(filename)
  return filename
end

function read_file(filename)
  local ifile = io.open(filename, "r")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return nil
  end

  local str = ifile:read("*a")
  if (not str) then
    iup.Message("Error", "Fail when reading from file: " .. filename)
    return nil
  end

  ifile:close()
  return str
```

```
end

function write_file(filename, str)
  local ifile = io.open(filename, "w")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return false
  end

  if (not ifile:write(str)) then
    iup.Message("Error", "Fail when writing to file: " .. filename)
  end

  ifile:close()
  return true
end

function new_file(ih)
  local dlg = iup.GetDialog(ih)
  local multitext = dlg.multitext

  dlg.title = "Untitled - Simple Notepad"
  multitext.filename = nil
  multitext.dirty = nil
  multitext.value = ""
end

function open_file(ih, filename)
  local str = read_file(filename)
  if (str) then
    local dlg = iup.GetDialog(ih)
    local multitext = dlg.multitext
    local config = multitext.config

    dlg.title = str_filetitle(filename).." - Simple Notepad"
    multitext.filename = filename
    multitext.dirty = nil
    multitext.value = str

    config:RecentUpdate(filename)
  end
end
```

```
function save_file(multitext)
  if (write_file(multitext.filename, multitext.value)) then
    multitext.dirty = nil
  end
end

function saveas_file(multitext, filename)
  if (write_file(filename, multitext.value)) then
    local dlg = iup.GetDialog(multitext)
    local config = multitext.config

    dlg.title = str_filetitle(filename)..” - Simple Notepad”
    multitext.filename = filename
    multitext.dirty = nil

    config:RecentUpdate(filename)
  end
end

function save_check(ih)
  local dlg = iup.GetDialog(ih)
  local multitext = dlg.multitext

  if (multitext.dirty) then
    local resp = iup.Alarm(”Warning”, ”File not saved! Save it now?”, ”Yes”, ”No”,
    if resp == 1 then -- save the changes and continue
      save_file(multitext)
    elseif resp == 3 then  -- cancel
      return false
    else  -- ignore the changes and continue
    end
  end
  return true
end


--******************************** Main (Part 1/2) ****************************


config = iup.config{}
config.app_name = ”simple_notepad”
```

```
config:Load()

lbl_statusbar = iup.label{title = "Lin 1, Col 1", expand = "HORIZONTAL", padding =

multitext = iup.text{
  multiline = "YES",
  expand = "YES",
  config = config,
  dirty = nil,
}

font = config:GetVariable("MainWindow", "Font")
if (font) then
  multitext.font = font
end

item_new = iup.item{title = "&New...\tCtrl+N", image = "IUP_FileNew"}
item_open = iup.item{title = "&Open...\tCtrl+O", image = "IUP_FileOpen"}
item_save = iup.item{title="&Save\tCtrl+S"}
item_saveas = iup.item{title="Save &As...", image = "IUP_FileSave"}
item_font = iup.item{title="&Font..."}
item_about = iup.item{title="&About..."}
item_find = iup.item{title="&Find...\tCtrl+F", image = "IUP_EditFind"}
item_goto = iup.item{title="&Go To..."}
item_copy = iup.item{title="&Copy\tCtrl+C", image = "IUP_EditCopy"}
item_paste = iup.item{title="&Paste\tCtrl+V", image = "IUP_EditPaste"}
item_cut = iup.item{title="Cu&t\tCtrl+X", image = "IUP_EditCut"}
item_delete = iup.item{title="&Delete\tDel", image = "IUP_EditErase"}
item_select_all = iup.item{title="Select &All\tCtrl+A"}
item_revert = iup.item{title="&Revert"}
item_exit = iup.item{title="E&xit"}


--********************************* Callbacks *********************************


function multitext:dropfiles_cb(filename)
  if (save_check(self)) then
    open_file(self, filename)
  end
end
```

```
function multitext:valuechanged_cb()
  self.dirty = "YES"
end

function config:recent_cb()
  if (save_check(self)) then
    local filename = self.title
    open_file(self, filename)
  end
end

function multitext:caret_cb(lin, col)
  lbl_statusbar.title = "Lin "..lin.." Col "..col
end

function item_new:action()
  if save_check(self) then
    new_file(self)
  end
end

function item_open:action()
  if not save_check(self) then
    return
  end

  local filedlg = iup.filedlg{
    dialogtype = "OPEN",
    filter = "*.txt",
    filterinfo = "Text Files",
    parentdialog=iup.GetDialog(self)
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    open_file(self, filename)
  end

  filedlg:destroy()
end
```

```
function item_saveas:action()
  local filedlg = iup.filedlg{
    dialogtype = "SAVE",
    filter = "*.txt",
    filterinfo = "Text Files",
    parentdialog = iup.GetDialog(self),
    file = multitext.filename,
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    saveas_file(multitext, filename)
  end

  filedlg:destroy()
end

function item_save:action()
  if (not multitext.filename) then
    item_saveas:action()
  else
    save_file(multitext)
  end
end

function item_revert:action()
  open_file(self, multitext.filename)
end

function item_exit:action()
  if not save_check(self) then
    return iup.IGNORE -- to abort the CLOSE_CB callback
  end

  config:DialogClosed(iup.GetDialog(self), "MainWindow")
  config:Save()
  config:destroy()
  return iup.CLOSE
end
```

```
function item_goto:action()
  local line_count = multitext.linecount
  local lbl_goto = iup.label{title = "Line Number [1-"..line_count.."]:"}
  local txt_goto = iup.text{mask = iup.MASK_UINT, visiblecolumns = 20} --unsigned i

  local bt_goto_ok = iup.button{title = "OK", text_linecount = 0, padding = "10x2"}
  bt_goto_ok.text_linecount = line_count
  function bt_goto_ok:action()
    local line_count = tonumber(self.text_linecount)
    local line = tonumber(txt_goto.value)
    if (line < 1 or line >= line_count) then
      iup.Message("Error", "Invalid line number.")
      return
    end
    goto_dlg.status = 1
    return iup.CLOSE
  end

  local bt_goto_cancel = iup.button{title = "Cancel", padding = "10x2"}
  function bt_goto_cancel:action()
    goto_dlg.status = 0
    return iup.CLOSE
  end

  local box = iup.vbox{
    lbl_goto,
    txt_goto,
    iup.hbox{
      iup.fill{},
      bt_goto_ok,
      bt_goto_cancel,
      normalizesize="HORIZONTAL",
    },
    margin = "10x10",
    gap = "5",
  }
  goto_dlg = iup.dialog{
    box,
    title = "Go To Line",
    dialogframe = "Yes",
    defaultenter = bt_goto_ok,
```

```
    defaultesc = bt_goto_cancel,
    parentdialog = iup.GetDialog(self)
  }

  goto_dlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(goto_dlg.status) == 1) then
    local line = txt_goto.value
    local pos = iup.TextConvertLinColToPos(multitext, line, 0)
    multitext.caretpos = pos
    multitext.scrolltopos = pos
  end

  goto_dlg:destroy()
end

function item_find:action()
  local find_dlg = self.find_dialog
  if (not find_dlg) then
    local find_txt = iup.text{visiblecolumns = "20"}
    local find_case = iup.toggle{title = "Case Sensitive"}
    local bt_find_next = iup.button{title = "Find Next", padding = "10x2"}
    local bt_find_close = iup.button{title = "Close", padding = "10x2"}

    function bt_find_next:action()
      local find_pos = tonumber(find_dlg.find_pos)
      local str_to_find = find_txt.value

      local casesensitive = (find_case.value == "ON")

      -- test again, because it can be called from the hot key
      if (not str_to_find or str_to_find:len()==0) then
        return
      end

      if (not find_pos) then
        find_pos = 1
      end

      local str = multitext.value

      local pos, end_pos = str_find(str, str_to_find, casesensitive, find_pos)
```

```
  if (not pos) then
    pos, end_pos = str_find(str, str_to_find, casesensitive, 1)  -- try again fr
  end

  if (pos) and (pos > 0) then
    pos = pos - 1
    find_dlg.find_pos = end_pos

    iup.SetFocus(multitext)
    multitext.selectionpos = pos..":"..end_pos

    local lin, col = iup.TextConvertPosToLinCol(multitext, pos)
    local pos = iup.TextConvertLinColToPos(multitext, lin, 0)  -- position at c
    multitext.scrolltopos = pos
  else
    find_dlg.find_pos = nil
    iup.Message("Warning", "Text not found.")
  end
end

function bt_find_close:action()
  iup.Hide(iup.GetDialog(self))  -- do not destroy, just hide
end

box = iup.vbox{
  iup.label{title = "Find What:"},
  find_txt,
  find_case,
  iup.hbox{
    iup.fill{},
    bt_find_next,
    bt_find_close,
    normalizesize="HORIZONTAL",
  },
  margin = "10x10",
  gap = "5",
}

find_dlg = iup.dialog{
  box,
  title = "Find",
  dialogframe = "Yes",
```

```
      defaultenter = bt_next,
      defaultesc = bt_close,
      parentdialog = iup.GetDialog(self)
      }

    -- Save the dialog to reuse it
    self.find_dialog = find_dlg -- from the main dialog */
  end

  -- centerparent first time, next time reuse the last position
  find_dlg:showxy(iup.CURRENT, iup.CURRENT)
end

function item_copy:action()
  local clipboard = iup.clipboard{text = multitext.selectedtext}
  clipboard:destroy()
end

function item_paste:action()
  local clipboard = iup.clipboard{}
  multitext.insert = clipboard.text
  clipboard:destroy()
  return iup.IGNORE -- avoid system processing for hot keys, to correctly parse li
end

function item_cut:action()
  local clipboard = iup.clipboard{text = multitext.selectedtext}
  multitext.selectedtext = ""
  clipboard:destroy()
end

function item_delete:action()
  multitext.selectedtext = ""
end

function item_select_all:action()
  iup.SetFocus(multitext)
  multitext.selection = "ALL"
end

function item_font:action()
  local font = multitext.font
```

```
  local fontdlg = iup.fontdlg{value = font, parentdialog=iup.GetDialog(self)}

  fontdlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(fontdlg.status) == 1) then
    multitext.font = fontdlg.value
    config:SetVariable("MainWindow", "Font", fontdlg.value)
  end

  fontdlg:destroy()
end

function item_about:action()
  iup.Message("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣
end


--********************************* Main (Part 2/2) ****************************


recent_menu = iup.menu{}

file_menu = iup.menu{
  item_new,
  item_open,
  item_save,
  item_saveas,
  item_revert,
  iup.separator{},
  iup.submenu{title="Recent␣&Files", recent_menu},
  item_exit
  }

function file_menu:open_cb()
  if (multitext.dirty) then
    item_save.active = "YES"
  else
    item_save.active = "NO"
  end
  if (multitext.dirty and multitext.filename) then
    item_revert.active = "YES"
  else
```

```
    item_revert.active = "NO"
  end
end

edit_menu = iup.menu{
    item_cut,
    item_copy,
    item_paste,
    item_delete,
    iup.separator{},
    item_find,
    item_goto,
    iup.separator{},
    item_select_all
    }

function edit_menu:open_cb()
  local clipboard = iup.clipboard{}

  if (not clipboard.textavailable) then
    item_paste.active = "NO"
  else
    item_paste.active = "YES"
  end

  if (not multitext.selectedtext) then
    item_cut.active = "NO"
    item_delete.active = "NO"
    item_copy.active = "NO"
  else
    item_cut.active = "YES"
    item_delete.active = "YES"
    item_copy.active = "YES"
  end

  clipboard:destroy()
end

format_menu = iup.menu{item_font}
help_menu = iup.menu{item_about}
sub_menu_file = iup.submenu{file_menu, title = "&File"}
sub_menu_edit = iup.submenu{edit_menu, title = "&Edit"}
```

```
sub_menu_format = iup.submenu{format_menu, title = "F&ormat"}
sub_menu_help = iup.submenu{help_menu, title = "&Help"}

menu = iup.menu{
  sub_menu_file,
  sub_menu_edit,
  sub_menu_format,
  sub_menu_help,
  }

btn_new = iup.button{image = "IUP_FileNew", flat = "Yes", action = item_new.action,
btn_open = iup.button{image = "IUP_FileOpen", flat = "Yes", action = item_open.actio
btn_save = iup.button{image = "IUP_FileSave", flat = "Yes", action = item_save.actio
btn_find = iup.button{image = "IUP_EditFind", flat = "Yes", action = item_find.actio
btn_cut = iup.button{image = "IUP_EditCut", flat = "Yes", action = item_cut.action,
btn_copy = iup.button{image =  "IUP_EditCopy", flat = "Yes", action = item_copy.act
btn_paste = iup.button{image = "IUP_EditPaste", flat = "Yes", action = item_paste.a

toolbar_hb = iup.hbox{
  btn_new,
  btn_open,
  btn_save,
  iup.label{separator="VERTICAL"},
  btn_cut,
  btn_copy,
  btn_paste,
  iup.label{separator="VERTICAL"},
  btn_find,
  margin = "5x5",
  gap = 2,
}

vbox = iup.vbox{
  toolbar_hb,
  multitext,
  lbl_statusbar,
}

dlg = iup.dialog{
  vbox,
  title = "Simple Notepad",
  size = "HALFxHALF",
```

```
  menu = menu,
  close_cb = item_exit.action,
  multitext = multitext,
  dropfiles_cb = multitext.dropfiles_cb,
}

function dlg:k_any(c)
  if (c == iup.K_cN) then
    item_new:action()
  elseif (c == iup.K_cO) then
    item_open:action()
  elseif (c == iup.K_cS) then
    item_save:action()
  elseif (c == iup.K_cF) then
    item_find:action()
  elseif (c == iup.K_cG) then
    item_goto:action()
  end
end

config:RecentInit(recent_menu, 10)

-- parent for pre-defined dialogs in closed functions (IupMessage)
iup.SetGlobal("PARENTDIALOG", iup.SetHandleName(dlg))

config:DialogShow(dlg, "MainWindow")

-- initialize the current file
new_file(dlg)

-- open a file from the command line (allow file association in Windows)
if (arg and arg[1]) then
  filename = arg[1]
  open_file(dlg, filename)
end

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

3.10 动态布局

现在，我们已经可以控制显示或隐藏对话框的元素，比如工具栏和状态栏。如果我们简单地设置它的 *VISIBLE* 属性值为 *NO*，它们会被隐藏，但是它们所占的空间仍被保留，被对话框背景替代。为了避免这种情况，我们可以设置 *VISIBLE* 属性的同时，使用 *FLOATING* 属性。这样，元素隐藏后，它的空间就被文本框占用。

为了演示这个功能，我们首先在主菜单中添加一个叫做 *View* 的子菜单，里面有两个菜单项，一个用来控制工具栏是否可见，一个用来控制状态栏是否可见。我们使用 *IupConfig* 来存储用户的选择，使得用户不用每次进行配置。

首先，我们来看 *toggle_bar_visiblity* 函数，它实际控制了我们的工具栏和状态栏的可见性。当 *View* 菜单下的菜单项被点击，如果对应的项目是可见的，就设置这个项目的 *FLOATING* 属性为 *YES*，*VISIBLE* 属性为 *NO*，然后设置菜单项的值为 *OFF*。如果对应的项目是不可见的，进行相反的操作即可。最后调用 *IupRefresh* 函数更新对话框布局。

接下来是两个新的回调，*item_toolbar_action_cb* 和 *item_statusbar_action_cb*。这两个回调负责调用 *toggle_bar_visibility* 和 *IupConfigSetVariableStr* 来存储菜单项的状态。

接下来需要修改的地方是主函数，在这里我们定义了 *View* 子菜单以及它的菜单项，最后绑定了它们的回调。

C语言

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <iup.h>
#include <iup_config.h>


/********************************** Utilities *************************************


const char* str_filetitle(const char *filename)
{
  /* Start at the last character */
  int len = (int)strlen(filename);
  int offset = len - 1;
  while (offset != 0)
  {
    if (filename[offset] == '\\' || filename[offset] == '/')
    {
      offset++;
      break;
    }
    offset--;
  }
```

```
  return filename + offset;
}

int str_compare(const char *l, const char *r, int casesensitive)
{
  if (!l || !r)
    return 0;

  while (*l && *r)
  {
    int diff;
    char l_char = *l,
      r_char = *r;

    /* compute the difference of both characters */
    if (casesensitive)
      diff = l_char - r_char;
    else
      diff = tolower((int)l_char) - tolower((int)r_char);

    /* if they differ we have a result */
    if (diff != 0)
      return 0;

    /* otherwise process the next characters */
    ++l;
    ++r;
  }

  /* check also for terminator */
  if (*l == *r)
    return 1;

  if (*r == 0)
    return 1;  /* if second string is at terminator, then it is partially equal */

  return 0;
}

int str_find(const char *str, const char *str_to_find, int casesensitive)
{
  int i, str_len, str_to_find_len, count;
```

```
  if (!str || str[0] == 0 || !str_to_find || str_to_find[0] == 0)
    return -1;

  str_len = (int)strlen(str);
  str_to_find_len = (int)strlen(str_to_find);
  count = str_len - str_to_find_len;
  if (count < 0)
    return -1;

  count++;

  for (i = 0; i<count; i++)
  {
    if (str_compare(str, str_to_find, casesensitive))
      return i;

    str++;
  }

  return -1;
}

char* read_file(const char* filename)
{
  int size;
  char* str;
  FILE* file = fopen(filename, "rb");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return NULL;
  }

  /* calculate file size */
  fseek(file, 0, SEEK_END);
  size = ftell(file);
  fseek(file, 0, SEEK_SET);

  /* allocate memory for the file contents + nul terminator */
  str = malloc(size + 1);
  /* read all data at once */
```

```
  fread(str, size, 1, file);
  /* set the nul terminator */
  str[size] = 0;

  if (ferror(file))
    IupMessagef("Error", "Fail when reading from file: %s", filename);

  fclose(file);
  return str;
}

int write_file(const char* filename, const char* str, int count)
{
  FILE* file = fopen(filename, "w");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return 0;
  }

  fwrite(str, 1, count, file);

  if (ferror(file))
    IupMessagef("Error", "Fail when writing to file: %s", filename);

  fclose(file);
  return 1;
}

void new_file(Ihandle* ih)
{
  Ihandle* dlg = IupGetDialog(ih);
  Ihandle* multitext = IupGetDialogChild(dlg, "MULTITEXT");

  IupSetAttribute(dlg, "TITLE", "Untitled - Simple Notepad");
  IupSetAttribute(multitext, "FILENAME", NULL);
  IupSetAttribute(multitext, "DIRTY", "NO");
  IupSetAttribute(multitext, "VALUE", "");
}

void open_file(Ihandle* ih, const char* filename)
{
```

```
  char* str = read_file(filename);
  if (str)
  {
    Ihandle* dlg = IupGetDialog(ih);
    Ihandle* multitext = IupGetDialogChild(dlg, "MULTITEXT");
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

    IupSetfAttribute(dlg, "TITLE", "%s - Simple Notepad", str_filetitle(filename));
    IupSetStrAttribute(multitext, "FILENAME", filename);
    IupSetAttribute(multitext, "DIRTY", "NO");
    IupSetStrAttribute(multitext, "VALUE", str);

    IupConfigRecentUpdate(config, filename);

    free(str);
  }
}

void save_file(Ihandle* multitext)
{
  char* filename = IupGetAttribute(multitext, "FILENAME");
  char* str = IupGetAttribute(multitext, "VALUE");
  int count = IupGetInt(multitext, "COUNT");
  if (write_file(filename, str, count))
    IupSetAttribute(multitext, "DIRTY", "NO");
}

void saveas_file(Ihandle* multitext, const char* filename)
{
  char* str = IupGetAttribute(multitext, "VALUE");
  int count = IupGetInt(multitext, "COUNT");
  if (write_file(filename, str, count))
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

    IupSetfAttribute(IupGetDialog(multitext), "TITLE", "%s - Simple Notepad", str_
    IupSetStrAttribute(multitext, "FILENAME", filename);
    IupSetAttribute(multitext, "DIRTY", "NO");

    IupConfigRecentUpdate(config, filename);
  }
}
```

```
int save_check(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");
  if (IupGetInt(multitext, "DIRTY"))
  {
    switch (IupAlarm("Warning", "File not saved! Save it now?", "Yes", "No", "Cance
    {
    case 1:  /* save the changes and continue */
      save_file(multitext);
      break;
    case 2:  /* ignore the changes and continue */
      break;
    case 3:  /* cancel */
      return 0;
    }
  }
  return 1;
}

void toggle_bar_visibility(Ihandle* item, Ihandle* ih)
{
  if (IupGetInt(item, "VALUE"))
  {
    IupSetAttribute(ih, "FLOATING", "YES");
    IupSetAttribute(ih, "VISIBLE", "NO");
    IupSetAttribute(item, "VALUE", "OFF");
  }
  else
  {
    IupSetAttribute(ih, "FLOATING", "NO");
    IupSetAttribute(ih, "VISIBLE", "YES");
    IupSetAttribute(item, "VALUE", "ON");
  }

  IupRefresh(ih);  /* refresh the dialog layout */
}


/********************************* Callbacks ************************************
```

```
int dropfiles_cb(Ihandle* ih, const char* filename)
{
  if (save_check(ih))
    open_file(ih, filename);

  return IUP_DEFAULT;
}

int multitext_valuechanged_cb(Ihandle* multitext)
{
  IupSetAttribute(multitext, "DIRTY", "YES");
  return IUP_DEFAULT;
}

int file_menu_open_cb(Ihandle* ih)
{
  Ihandle* item_revert = IupGetDialogChild(ih, "ITEM_REVERT");
  Ihandle* item_save = IupGetDialogChild(ih, "ITEM_SAVE");
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");
  char* filename = IupGetAttribute(multitext, "FILENAME");
  int dirty = IupGetInt(multitext, "DIRTY");

  if (dirty)
    IupSetAttribute(item_save, "ACTIVE", "YES");
  else
    IupSetAttribute(item_save, "ACTIVE", "NO");

  if (dirty && filename)
    IupSetAttribute(item_revert, "ACTIVE", "YES");
  else
    IupSetAttribute(item_revert, "ACTIVE", "NO");
  return IUP_DEFAULT;
}

int edit_menu_open_cb(Ihandle* ih)
{
  Ihandle *clipboard = IupClipboard();

  Ihandle *item_paste = IupGetDialogChild(ih, "ITEM_PASTE");
  Ihandle *item_cut = IupGetDialogChild(ih, "ITEM_CUT");
  Ihandle *item_delete = IupGetDialogChild(ih, "ITEM_DELETE");
  Ihandle *item_copy = IupGetDialogChild(ih, "ITEM_COPY");
```

```
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  if (!IupGetInt(clipboard, "TEXTAVAILABLE"))
    IupSetAttribute(item_paste, "ACTIVE", "NO");
  else
    IupSetAttribute(item_paste, "ACTIVE", "YES");

  if (!IupGetAttribute(multitext, "SELECTEDTEXT"))
  {
    IupSetAttribute(item_cut, "ACTIVE", "NO");
    IupSetAttribute(item_delete, "ACTIVE", "NO");
    IupSetAttribute(item_copy, "ACTIVE", "NO");
  }
  else
  {
    IupSetAttribute(item_cut, "ACTIVE", "YES");
    IupSetAttribute(item_delete, "ACTIVE", "YES");
    IupSetAttribute(item_copy, "ACTIVE", "YES");
  }

  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int config_recent_cb(Ihandle* ih)
{
  if (save_check(ih))
  {
    char* filename = IupGetAttribute(ih, "TITLE");
    open_file(ih, filename);
  }
  return IUP_DEFAULT;
}

int multitext_caret_cb(Ihandle *ih, int lin, int col)
{
  Ihandle *lbl_statusbar = IupGetDialogChild(ih, "STATUSBAR");
  IupSetfAttribute(lbl_statusbar, "TITLE", "Lin %d, Col %d", lin, col);
  return IUP_DEFAULT;
}

int item_new_action_cb(Ihandle* item_new)
```

```
{
  if (save_check(item_new))
    new_file(item_new);

  return IUP_DEFAULT;
}

int item_open_action_cb(Ihandle* item_open)
{
  Ihandle *filedlg;

  if (!save_check(item_open))
    return IUP_DEFAULT;

  filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "OPEN");
  IupSetAttribute(filedlg, "EXTFILTER", "Text␣Files|*.txt|All␣Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_open));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    open_file(item_open, filename);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_saveas_action_cb(Ihandle* item_saveas)
{
  Ihandle* multitext = IupGetDialogChild(item_saveas, "MULTITEXT");
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "SAVE");
  IupSetAttribute(filedlg, "EXTFILTER", "Text␣Files|*.txt|All␣Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_saveas));
  IupSetStrAttribute(filedlg, "FILE", IupGetAttribute(multitext, "FILENAME"));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(filedlg, "STATUS") != -1)
```

```
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    saveas_file(multitext, filename);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_save_action_cb(Ihandle* item_save)
{
  Ihandle* multitext = IupGetDialogChild(item_save, "MULTITEXT");
  char* filename = IupGetAttribute(multitext, "FILENAME");
  if (!filename)
    item_saveas_action_cb(item_save);
  else
    save_file(multitext);
  return IUP_DEFAULT;
}

int item_revert_action_cb(Ihandle* item_revert)
{
  Ihandle* multitext = IupGetDialogChild(item_revert, "MULTITEXT");
  char* filename = IupGetAttribute(multitext, "FILENAME");
  open_file(item_revert, filename);
  return IUP_DEFAULT;
}

int item_exit_action_cb(Ihandle* item_exit)
{
  Ihandle* dlg = IupGetDialog(item_exit);
  Ihandle* config = (Ihandle*)IupGetAttribute(dlg, "CONFIG");

  if (!save_check(item_exit))
    return IUP_IGNORE;  /* to abort the CLOSE_CB callback */

  IupConfigDialogClosed(config, dlg, "MainWindow");
  IupConfigSave(config);
  IupDestroy(config);
  return IUP_CLOSE;
}
```

```
int goto_ok_action_cb(Ihandle* bt_ok)
{
  int line_count = IupGetInt(bt_ok, "TEXT_LINECOUNT");
  Ihandle* txt = IupGetDialogChild(bt_ok, "LINE_TEXT");
  int line = IupGetInt(txt, "VALUE");
  if (line < 1 || line >= line_count)
  {
    IupMessage("Error", "Invalid line number.");
    return IUP_DEFAULT;
  }

  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "1");
  return IUP_CLOSE;
}

int goto_cancel_action_cb(Ihandle* bt_ok)
{
  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "0");
  return IUP_CLOSE;
}

int item_goto_action_cb(Ihandle* item_goto)
{
  Ihandle* multitext = IupGetDialogChild(item_goto, "MULTITEXT");
  Ihandle *dlg, *box, *bt_ok, *bt_cancel, *txt, *lbl;
  int line_count = IupGetInt(multitext, "LINECOUNT");

  lbl = IupLabel(NULL);
  IupSetfAttribute(lbl, "TITLE", "Line Number [1-%d]:", line_count);
  txt = IupText(NULL);
  IupSetAttribute(txt, "MASK", IUP_MASK_UINT);  /* unsigned integer numbers only */
  IupSetAttribute(txt, "NAME", "LINE_TEXT");
  IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
  bt_ok = IupButton("OK", NULL);
  IupSetInt(bt_ok, "TEXT_LINECOUNT", line_count);
  IupSetAttribute(bt_ok, "PADDING", "10x2");
  IupSetCallback(bt_ok, "ACTION", (Icallback)goto_ok_action_cb);
  bt_cancel = IupButton("Cancel", NULL);
  IupSetCallback(bt_cancel, "ACTION", (Icallback)goto_cancel_action_cb);
  IupSetAttribute(bt_cancel, "PADDING", "10x2");

  box = IupVbox(
```

```
    lbl,
    txt,
    IupSetAttributes(IupHbox(
      IupFill(),
      bt_ok,
      bt_cancel,
      NULL), "NORMALIZESIZE=HORIZONTAL"),
    NULL);
  IupSetAttribute(box, "MARGIN", "10x10");
  IupSetAttribute(box, "GAP", "5");

  dlg = IupDialog(box);
  IupSetAttribute(dlg, "TITLE", "Go To Line");
  IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
  IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_ok);
  IupSetAttributeHandle(dlg, "DEFAULTESC", bt_cancel);
  IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_goto));

  IupPopup(dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(dlg, "STATUS") == 1)
  {
    int line = IupGetInt(txt, "VALUE");
    int pos;
    IupTextConvertLinColToPos(multitext, line, 0, &pos);
    IupSetInt(multitext, "CARETPOS", pos);
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }

  IupDestroy(dlg);

  return IUP_DEFAULT;
}

int find_next_action_cb(Ihandle* bt_next)
{
  Ihandle* multitext = (Ihandle*)IupGetAttribute(bt_next, "MULTITEXT");
  char* str = IupGetAttribute(multitext, "VALUE");
  int find_pos = IupGetInt(multitext, "FIND_POS");

  Ihandle* txt = IupGetDialogChild(bt_next, "FIND_TEXT");
  char* str_to_find = IupGetAttribute(txt, "VALUE");
```

```
  Ihandle* find_case = IupGetDialogChild(bt_next, "FIND_CASE");
  int casesensitive = IupGetInt(find_case, "VALUE");

  int pos = str_find(str + find_pos, str_to_find, casesensitive);
  if (pos >= 0)
    pos += find_pos;
  else if (find_pos > 0)
    pos = str_find(str, str_to_find, casesensitive);  /* try again from the start *

  if (pos >= 0)
  {
    int lin, col,
      end_pos = pos + (int)strlen(str_to_find);

    IupSetInt(multitext, "FIND_POS", end_pos);

    IupSetFocus(multitext);
    IupSetfAttribute(multitext, "SELECTIONPOS", "%d:%d", pos, end_pos);

    IupTextConvertPosToLinCol(multitext, pos, &lin, &col);
    IupTextConvertLinColToPos(multitext, lin, 0, &pos);  /* position at col=0, just
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }
  else
    IupMessage("Warning", "Text not found.");

  return IUP_DEFAULT;
}

int find_close_action_cb(Ihandle* bt_close)
{
  IupHide(IupGetDialog(bt_close));
  return IUP_DEFAULT;
}

int item_find_action_cb(Ihandle* item_find)
{
  Ihandle* dlg = (Ihandle*)IupGetAttribute(item_find, "FIND_DIALOG");
  if (!dlg)
  {
    Ihandle* multitext = IupGetDialogChild(item_find, "MULTITEXT");
```

```
  Ihandle *box, *bt_next, *bt_close, *txt, *find_case;

  txt = IupText(NULL);
  IupSetAttribute(txt, "NAME", "FIND_TEXT");
  IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
  find_case = IupToggle("Case Sensitive", NULL);
  IupSetAttribute(find_case, "NAME", "FIND_CASE");
  bt_next = IupButton("Find Next", NULL);
  IupSetAttribute(bt_next, "PADDING", "10x2");
  IupSetCallback(bt_next, "ACTION", (Icallback)find_next_action_cb);
  bt_close = IupButton("Close", NULL);
  IupSetCallback(bt_close, "ACTION", (Icallback)find_close_action_cb);
  IupSetAttribute(bt_close, "PADDING", "10x2");

  box = IupVbox(
    IupLabel("Find What:"),
    txt,
    find_case,
    IupSetAttributes(IupHbox(
      IupFill(),
      bt_next,
      bt_close,
      NULL), "NORMALIZESIZE=HORIZONTAL"),
    NULL);
  IupSetAttribute(box, "MARGIN", "10x10");
  IupSetAttribute(box, "GAP", "5");

  dlg = IupDialog(box);
  IupSetAttribute(dlg, "TITLE", "Find");
  IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
  IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_next);
  IupSetAttributeHandle(dlg, "DEFAULTESC", bt_close);
  IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_find));

  /* Save the multiline to acess it from the callbacks */
  IupSetAttribute(dlg, "MULTITEXT", (char*)multitext);

  /* Save the dialog to reuse it */
  IupSetAttribute(item_find, "FIND_DIALOG", (char*)dlg);
}

/* centerparent first time, next time reuse the last position */
```

```
  IupShowXY(dlg, IUP_CURRENT, IUP_CURRENT);

  return IUP_DEFAULT;
}

int item_copy_action_cb(Ihandle* item_copy)
{
  Ihandle* multitext = IupGetDialogChild(item_copy, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(clipboard, "TEXT", IupGetAttribute(multitext, "SELECTEDTEXT"));
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_paste_action_cb(Ihandle* item_paste)
{
  Ihandle* multitext = IupGetDialogChild(item_paste, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(multitext, "INSERT", IupGetAttribute(clipboard, "TEXT"));
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_cut_action_cb(Ihandle* item_cut)
{
  Ihandle* multitext = IupGetDialogChild(item_cut, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(clipboard, "TEXT", IupGetAttribute(multitext, "SELECTEDTEXT"));
  IupSetAttribute(multitext, "SELECTEDTEXT", "");
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_delete_action_cb(Ihandle* item_delete)
{
  Ihandle* multitext = IupGetDialogChild(item_delete, "MULTITEXT");
  IupSetAttribute(multitext, "SELECTEDTEXT", "");
  return IUP_DEFAULT;
}

int item_select_all_action_cb(Ihandle* item_select_all)
{
```

```
  Ihandle* multitext = IupGetDialogChild(item_select_all, "MULTITEXT");
  IupSetFocus(multitext);
  IupSetAttribute(multitext, "SELECTION", "ALL");
  return IUP_DEFAULT;
}

int item_font_action_cb(Ihandle* item_font)
{
  Ihandle* multitext = IupGetDialogChild(item_font, "MULTITEXT");
  Ihandle* fontdlg = IupFontDlg();
  char* font = IupGetAttribute(multitext, "FONT");
  IupSetStrAttribute(fontdlg, "VALUE", font);
  IupSetAttributeHandle(fontdlg, "PARENTDIALOG", IupGetDialog(item_font));

  IupPopup(fontdlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(fontdlg, "STATUS") == 1)
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
    font = IupGetAttribute(fontdlg, "VALUE");
    IupSetStrAttribute(multitext, "FONT", font);

    IupConfigSetVariableStr(config, "MainWindow", "Font", font);
  }

  IupDestroy(fontdlg);
  return IUP_DEFAULT;
}

int item_toolbar_action_cb(Ihandle* item_toolbar)
{
  Ihandle* multitext = IupGetDialogChild(item_toolbar, "MULTITEXT");
  Ihandle* toolbar = IupGetChild(IupGetParent(multitext), 0);
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

  toggle_bar_visibility(item_toolbar, toolbar);

  IupConfigSetVariableStr(config, "MainWindow", "Toolbar", IupGetAttribute(item_too
  return IUP_DEFAULT;
}

int item_statusbar_action_cb(Ihandle* item_statusbar)
```

```
{
  Ihandle* multitext = IupGetDialogChild(item_statusbar, "MULTITEXT");
  Ihandle* statusbar = IupGetBrother(multitext);
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

  toggle_bar_visibility(item_statusbar, statusbar);

  IupConfigSetVariableStr(config, "MainWindow", "Statusbar", IupGetAttribute(item_s
  return IUP_DEFAULT;
}

int item_about_action_cb(void)
{
  IupMessage("About", "   Simple Notepad\n\nAutors:\n   Gustavo Lyrio\n   Antonio S
  return IUP_DEFAULT;
}


/********************************* Main *****************************************/


int main(int argc, char **argv)
{
  Ihandle *dlg, *vbox, *multitext, *menu;
  Ihandle *sub_menu_file, *file_menu, *item_exit, *item_new, *item_open, *item_save
  Ihandle *sub_menu_edit, *edit_menu, *item_find, *item_goto, *item_copy, *item_pas
  Ihandle *btn_cut, *btn_copy, *btn_paste, *btn_find, *btn_new, *btn_open, *btn_sav
  Ihandle *sub_menu_format, *format_menu, *item_font;
  Ihandle *sub_menu_help, *help_menu, *item_about;
  Ihandle *sub_menu_view, *view_menu, *item_toolbar, *item_statusbar;
  Ihandle *lbl_statusbar, *toolbar_hb, *recent_menu;
  Ihandle *config;
  const char* font;

  IupOpen(&argc, &argv);
  IupImageLibOpen();

  config = IupConfig();
  IupSetAttribute(config, "APP_NAME", "simple_notepad");
  IupConfigLoad(config);

  multitext = IupText(NULL);
```

```
  IupSetAttribute(multitext, "MULTILINE", "YES");
  IupSetAttribute(multitext, "EXPAND", "YES");
  IupSetAttribute(multitext, "NAME", "MULTITEXT");
  IupSetAttribute(multitext, "DIRTY", "NO");
  IupSetCallback(multitext, "CARET_CB", (Icallback)multitext_caret_cb);
  IupSetCallback(multitext, "VALUECHANGED_CB", (Icallback)multitext_valuechanged_
  IupSetCallback(multitext, "DROPFILES_CB", (Icallback)dropfiles_cb);

font = IupConfigGetVariableStr(config, "MainWindow", "Font");
if (font)
  IupSetStrAttribute(multitext, "FONT", font);

lbl_statusbar = IupLabel("Lin 1, Col 1");
  IupSetAttribute(lbl_statusbar, "NAME", "STATUSBAR");
  IupSetAttribute(lbl_statusbar, "EXPAND", "HORIZONTAL");
  IupSetAttribute(lbl_statusbar, "PADDING", "10x5");

item_new = IupItem("New\tCtrl+N", NULL);
  IupSetAttribute(item_new, "IMAGE", "IUP_FileNew");
  IupSetCallback(item_new, "ACTION", (Icallback)item_new_action_cb);
btn_new = IupButton(NULL, NULL);
  IupSetAttribute(btn_new, "IMAGE", "IUP_FileNew");
  IupSetAttribute(btn_new, "FLAT", "Yes");
  IupSetCallback(btn_new, "ACTION", (Icallback)item_new_action_cb);
  IupSetAttribute(btn_new, "TIP", "New (Ctrl+N)");
  IupSetAttribute(btn_new, "CANFOCUS", "No");

item_open = IupItem("&Open...\tCtrl+O", NULL);
  IupSetAttribute(item_open, "IMAGE", "IUP_FileOpen");
  IupSetCallback(item_open, "ACTION", (Icallback)item_open_action_cb);
btn_open = IupButton(NULL, NULL);
  IupSetAttribute(btn_open, "IMAGE", "IUP_FileOpen");
  IupSetAttribute(btn_open, "FLAT", "Yes");
  IupSetCallback(btn_open, "ACTION", (Icallback)item_open_action_cb);
  IupSetAttribute(btn_open, "TIP", "Open (Ctrl+O)");
  IupSetAttribute(btn_open, "CANFOCUS", "No");

item_save = IupItem("Save\tCtrl+S", NULL);
  IupSetAttribute(item_save, "NAME", "ITEM_SAVE");
  IupSetAttribute(item_save, "IMAGE", "IUP_FileSave");
  IupSetCallback(item_save, "ACTION", (Icallback)item_save_action_cb);
btn_save = IupButton(NULL, NULL);
```

```
  IupSetAttribute(btn_save, "IMAGE", "IUP_FileSave");
  IupSetAttribute(btn_save, "FLAT", "Yes");
  IupSetCallback(btn_save, "ACTION", (Icallback)item_save_action_cb);
  IupSetAttribute(btn_save, "TIP", "Save (Ctrl+S)");
  IupSetAttribute(btn_save, "CANFOCUS", "No");

item_saveas = IupItem("Save &As...", NULL);
  IupSetAttribute(item_saveas, "NAME", "ITEM_SAVEAS");
  IupSetCallback(item_saveas, "ACTION", (Icallback)item_saveas_action_cb);

item_revert = IupItem("Revert", NULL);
  IupSetAttribute(item_revert, "NAME", "ITEM_REVERT");
  IupSetCallback(item_revert, "ACTION", (Icallback)item_revert_action_cb);

item_exit = IupItem("E&xit", NULL);
  IupSetCallback(item_exit, "ACTION", (Icallback)item_exit_action_cb);

item_find = IupItem("&Find...\tCtrl+F", NULL);
  IupSetAttribute(item_find, "IMAGE", "IUP_EditFind");
  IupSetCallback(item_find, "ACTION", (Icallback)item_find_action_cb);
btn_find = IupButton(NULL, NULL);
  IupSetAttribute(btn_find, "IMAGE", "IUP_EditFind");
  IupSetAttribute(btn_find, "FLAT", "Yes");
  IupSetCallback(btn_find, "ACTION", (Icallback)item_find_action_cb);
  IupSetAttribute(btn_find, "TIP", "Find (Ctrl+F)");
  IupSetAttribute(btn_find, "CANFOCUS", "No");

item_cut = IupItem("Cut\tCtrl+X", NULL);
  IupSetAttribute(item_cut, "NAME", "ITEM_CUT");
  IupSetAttribute(item_cut, "IMAGE", "IUP_EditCut");
  IupSetCallback(item_cut, "ACTION", (Icallback)item_cut_action_cb);
item_copy = IupItem("Copy\tCtrl+C", NULL);
  IupSetAttribute(item_copy, "NAME", "ITEM_COPY");
  IupSetAttribute(item_copy, "IMAGE", "IUP_EditCopy");
  IupSetCallback(item_copy, "ACTION", (Icallback)item_copy_action_cb);
item_paste = IupItem("Paste\tCtrl+V", NULL);
  IupSetAttribute(item_paste, "NAME", "ITEM_PASTE");
  IupSetAttribute(item_paste, "IMAGE", "IUP_EditPaste");
  IupSetCallback(item_paste, "ACTION", (Icallback)item_paste_action_cb);
item_delete = IupItem("Delete\tDel", NULL);
  IupSetAttribute(item_delete, "IMAGE", "IUP_EditErase");
  IupSetAttribute(item_delete, "NAME", "ITEM_DELETE");
```

```
  IupSetCallback(item_delete, "ACTION", (Icallback)item_delete_action_cb);
item_select_all = IupItem("Select All\tCtrl+A", NULL);
  IupSetCallback(item_select_all, "ACTION", (Icallback)item_select_all_action_cb)

btn_cut = IupButton(NULL, NULL);
  IupSetAttribute(btn_cut, "IMAGE", "IUP_EditCut");
  IupSetAttribute(btn_cut, "FLAT", "Yes");
  IupSetCallback(btn_cut, "ACTION", (Icallback)item_cut_action_cb);
btn_copy = IupButton(NULL, NULL);
  IupSetAttribute(btn_copy, "IMAGE", "IUP_EditCopy");
  IupSetAttribute(btn_copy, "FLAT", "Yes");
  IupSetCallback(btn_copy, "ACTION", (Icallback)item_copy_action_cb);
btn_paste = IupButton(NULL, NULL);
  IupSetAttribute(btn_paste, "IMAGE", "IUP_EditPaste");
  IupSetAttribute(btn_paste, "FLAT", "Yes");
  IupSetCallback(btn_paste, "ACTION", (Icallback)item_paste_action_cb);

toolbar_hb = IupHbox(
  btn_new,
  btn_open,
  btn_save,
  IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
  btn_cut,
  btn_copy,
  btn_paste,
  IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
  btn_find,
  NULL);
IupSetAttribute(toolbar_hb, "MARGIN", "5x5");
IupSetAttribute(toolbar_hb, "GAP", "2");

item_toolbar = IupItem("&Toobar", NULL);
IupSetCallback(item_toolbar, "ACTION", (Icallback)item_toolbar_action_cb);
IupSetAttribute(item_toolbar, "VALUE", "ON");
item_statusbar = IupItem("&Statusbar", NULL);
IupSetCallback(item_statusbar, "ACTION", (Icallback)item_statusbar_action_cb);
IupSetAttribute(item_statusbar, "VALUE", "ON");

if (!IupConfigGetVariableIntDef(config, "MainWindow", "Toolbar", 1))
{
  IupSetAttribute(item_toolbar, "VALUE", "OFF");
```

```
    IupSetAttribute(toolbar_hb, "FLOATING", "YES");
    IupSetAttribute(toolbar_hb, "VISIBLE", "NO");
  }

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Statusbar", 1))
  {
    IupSetAttribute(item_statusbar, "VALUE", "OFF");

    IupSetAttribute(lbl_statusbar, "FLOATING", "YES");
    IupSetAttribute(lbl_statusbar, "VISIBLE", "NO");
  }

  item_goto = IupItem("&Go To...\tCtrl+G", NULL);
    IupSetCallback(item_goto, "ACTION", (Icallback)item_goto_action_cb);

  item_font = IupItem("&Font...", NULL);
    IupSetCallback(item_font, "ACTION", (Icallback)item_font_action_cb);
  item_about = IupItem("&About...", NULL);
    IupSetCallback(item_about, "ACTION", (Icallback)item_about_action_cb);

  recent_menu = IupMenu(NULL);

  file_menu = IupMenu(
    item_new,
    item_open,
    item_save,
    item_saveas,
    item_revert,
    IupSeparator(),
    IupSubmenu("Recent &Files", recent_menu),
    item_exit,
    NULL);
  edit_menu = IupMenu(
    item_cut,
    item_copy,
    item_paste,
    item_delete,
    IupSeparator(),
    item_find,
    item_goto,
    IupSeparator(),
    item_select_all,
```

```
  NULL);
format_menu = IupMenu(
  item_font,
  NULL);
view_menu = IupMenu(
  item_toolbar,
  item_statusbar,
  NULL);
help_menu = IupMenu(
  item_about,
  NULL);

IupSetCallback(file_menu, "OPEN_CB", (Icallback)file_menu_open_cb);
IupSetCallback(edit_menu, "OPEN_CB", (Icallback)edit_menu_open_cb);

sub_menu_file = IupSubmenu("&File", file_menu);
sub_menu_edit = IupSubmenu("&Edit", edit_menu);
sub_menu_format = IupSubmenu("F&ormat", format_menu);
sub_menu_view = IupSubmenu("&View", view_menu);
sub_menu_help = IupSubmenu("&Help", help_menu);

menu = IupMenu(
  sub_menu_file,
  sub_menu_edit,
  sub_menu_format,
  sub_menu_view,
  sub_menu_help,
  NULL);

vbox = IupVbox(
  toolbar_hb,
  multitext,
  lbl_statusbar,
  NULL);

dlg = IupDialog(vbox);
IupSetAttributeHandle(dlg, "MENU", menu);
IupSetAttribute(dlg, "SIZE", "HALFxHALF");
IupSetCallback(dlg, "CLOSE_CB", (Icallback)item_exit_action_cb);
IupSetCallback(dlg, "DROPFILES_CB", (Icallback)dropfiles_cb);

IupSetAttribute(dlg, "CONFIG", (char*)config);
```

```
  /* parent for pre-defined dialogs in closed functions (IupMessage) */
  IupSetAttributeHandle(NULL, "PARENTDIALOG", dlg);

  IupSetCallback(dlg, "K_cN", (Icallback)item_new_action_cb);
  IupSetCallback(dlg, "K_cO", (Icallback)item_open_action_cb);
  IupSetCallback(dlg, "K_cS", (Icallback)item_save_action_cb);
  IupSetCallback(dlg, "K_cF", (Icallback)item_find_action_cb);
  IupSetCallback(dlg, "K_cG", (Icallback)item_goto_action_cb);

  IupConfigRecentInit(config, recent_menu, config_recent_cb, 10);

  IupConfigDialogShow(config, dlg, "MainWindow");

  /* initialize the current file */
  new_file(dlg);

  /* open a file from the command line (allow file association in Windows) */
  if (argc > 1 && argv[1])
  {
    const char* filename = argv[1];
    open_file(dlg, filename);
  }

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")
require("iupluaimglib")


--********************************* Utilities *********************************


function str_find(str, str_to_find, casesensitive, start)
  if (not casesensitive) then
    return str_find(string.lower(str), string.lower(str_to_find), true, start)
  end
```

```
  return string.find(str, str_to_find, start, true)
end

function str_filetitle(filename)
  local filename = string.gsub(filename, "\\", "/")
  filename = string.reverse(filename)
  final = string.find(filename, '/')
  filename = string.sub(filename, 1, final-1)
  filename = string.reverse(filename)
  return filename
end

function read_file(filename)
  local ifile = io.open(filename, "r")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return nil
  end

  local str = ifile:read("*a")
  if (not str) then
    iup.Message("Error", "Fail when reading from file: " .. filename)
    return nil
  end

  ifile:close()
  return str
end

function write_file(filename, str)
  local ifile = io.open(filename, "w")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return false
  end

  if (not ifile:write(str)) then
    iup.Message("Error", "Fail when writing to file: " .. filename)
  end

  ifile:close()
```

```
  return true
end

function new_file(ih)
  local dlg = iup.GetDialog(ih)
  local multitext = dlg.multitext

  dlg.title = "Untitled - Simple Notepad"
  multitext.filename = nil
  multitext.dirty = nil
  multitext.value = ""
end

function open_file(ih, filename)
  local str = read_file(filename)
  if (str) then
    local dlg = iup.GetDialog(ih)
    local multitext = dlg.multitext
    local config = multitext.config

    dlg.title = str_filetitle(filename).." - Simple Notepad"
    multitext.filename = filename
    multitext.dirty = nil
    multitext.value = str

    config:RecentUpdate(filename)
  end
end

function save_file(multitext)
  if (write_file(multitext.filename, multitext.value)) then
    multitext.dirty = nil
  end
end

function saveas_file(multitext, filename)
  if (write_file(filename, multitext.value)) then
    local dlg = iup.GetDialog(multitext)
    local config = multitext.config

    dlg.title = str_filetitle(filename).." - Simple Notepad"
    multitext.filename = filename
```

```
    multitext.dirty = nil

    config:RecentUpdate(filename)
  end
end

function save_check(ih)
  local dlg = iup.GetDialog(ih)
  local multitext = dlg.multitext

  if (multitext.dirty) then
    local resp = iup.Alarm("Warning", "File not saved! Save it now?", "Yes", "No",
    if resp == 1 then -- save the changes and continue
      save_file(multitext)
    elseif resp == 3 then  -- cancel
      return false
    else  -- ignore the changes and continue
    end
  end
  return true
end

function toggle_bar_visibility(item, bar)
  if (item.value == "ON") then
    bar.floating = "YES"
    bar.visible = "NO"
    item.value = "OFF"
  else
    bar.floating = "NO"
    bar.visible = "YES"
    item.value = "ON"
  end
  iup.Refresh(bar)  -- refresh the dialog layout
end


--******************************** Main (Part 1/2) ****************************


config = iup.config{}
config.app_name = "simple_notepad"
config:Load()
```

```
lbl_statusbar = iup.label{title = "Lin 1, Col 1", expand = "HORIZONTAL", padding =

multitext = iup.text{
  multiline = "YES",
  expand = "YES",
  config = config,
  dirty = nil,
}

font = config:GetVariable("MainWindow", "Font")
if (font) then
  multitext.font = font
end

item_new = iup.item{title = "&New...\tCtrl+N", image = "IUP_FileNew"}
item_open = iup.item{title = "&Open...\tCtrl+O", image = "IUP_FileOpen"}
item_save = iup.item{title="&Save\tCtrl+S"}
item_saveas = iup.item{title="Save &As...", image = "IUP_FileSave"}
item_font = iup.item{title="&Font..."}
item_about = iup.item{title="&About..."}
item_find = iup.item{title="&Find...\tCtrl+F", image = "IUP_EditFind"}
item_goto = iup.item{title="&Go To..."}
item_copy = iup.item{title="&Copy\tCtrl+C", image = "IUP_EditCopy"}
item_paste = iup.item{title="&Paste\tCtrl+V", image = "IUP_EditPaste"}
item_cut = iup.item{title="Cu&t\tCtrl+X", image = "IUP_EditCut"}
item_delete = iup.item{title="&Delete\tDel", image = "IUP_EditErase"}
item_select_all = iup.item{title="Select &All\tCtrl+A"}
item_revert = iup.item{title="&Revert"}
item_exit = iup.item{title="E&xit"}
item_toolbar = iup.item{title="&Toobar", value="ON"}
item_statusbar = iup.item{title="&Statusbar", value="ON"}

show_statusbar = config:GetVariableDef("MainWindow", "Statusbar", "ON")
if (show_statusbar == "OFF") then
  item_statusbar.value = "OFF"
  lbl_statusbar.floating = "YES"
  lbl_statusbar.visible = "NO"
end


--******************************** Callbacks ********************************
```

```
function multitext:dropfiles_cb(filename)
  if (save_check(self)) then
    open_file(self, filename)
  end
end

function multitext:valuechanged_cb()
  self.dirty = "YES"
end

function config:recent_cb()
  if (save_check(self)) then
    local filename = self.title
    open_file(self, filename)
  end
end

function multitext:caret_cb(lin, col)
  lbl_statusbar.title = "Lin "..lin.."," Col "..col
end

function item_new:action()
  if save_check(self) then
    new_file(self)
  end
end

function item_open:action()
  if not save_check(self) then
    return
  end

  local filedlg = iup.filedlg{
    dialogtype = "OPEN",
    filter = "*.txt",
    filterinfo = "Text Files",
    parentdialog=iup.GetDialog(self)
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)
```

```
  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    open_file(self, filename)
  end

  filedlg:destroy()
end

function item_saveas:action()
  local filedlg = iup.filedlg{
    dialogtype = "SAVE",
    filter = "*.txt",
    filterinfo = "Text Files",
    parentdialog = iup.GetDialog(self),
    file = multitext.filename,
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    saveas_file(multitext, filename)
  end

  filedlg:destroy()
end

function item_save:action()
  if (not multitext.filename) then
    item_saveas:action()
  else
    save_file(multitext)
  end
end

function item_revert:action()
  open_file(self, multitext.filename)
end

function item_exit:action()
  if not save_check(self) then
```

```
    return iup.IGNORE  -- to abort the CLOSE_CB callback
  end

  config:DialogClosed(iup.GetDialog(self), "MainWindow")
  config:Save()
  config:destroy()
  return iup.CLOSE
end

function item_goto:action()
  local line_count = multitext.linecount
  local lbl_goto = iup.label{title = "Line Number [1-"..line_count.."]:"}
  local txt_goto = iup.text{mask = iup.MASK_UINT, visiblecolumns = 20} --unsigned i

  local bt_goto_ok = iup.button{title = "OK", text_linecount = 0, padding = "10x2"}
  bt_goto_ok.text_linecount = line_count
  function bt_goto_ok:action()
    local line_count = tonumber(self.text_linecount)
    local line = tonumber(txt_goto.value)
    if (line < 1 or line >= line_count) then
      iup.Message("Error", "Invalid line number.")
      return
    end
    goto_dlg.status = 1
    return iup.CLOSE
  end

  local bt_goto_cancel = iup.button{title = "Cancel", padding = "10x2"}
  function bt_goto_cancel:action()
    goto_dlg.status = 0
    return iup.CLOSE
  end

  local box = iup.vbox{
    lbl_goto,
    txt_goto,
    iup.hbox{
      iup.fill{},
      bt_goto_ok,
      bt_goto_cancel,
      normalizesize="HORIZONTAL",
    },
```

```
    margin = "10x10",
    gap = "5",
  }
  goto_dlg = iup.dialog{
    box,
    title = "Go To Line",
    dialogframe = "Yes",
    defaultenter = bt_goto_ok,
    defaultesc = bt_goto_cancel,
    parentdialog = iup.GetDialog(self)
  }

  goto_dlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(goto_dlg.status) == 1) then
    local line = txt_goto.value
    local pos = iup.TextConvertLinColToPos(multitext, line, 0)
    multitext.caretpos = pos
    multitext.scrolltopos = pos
  end

  goto_dlg:destroy()
end

function item_find:action()
  local find_dlg = self.find_dialog
  if (not find_dlg) then
    local find_txt = iup.text{visiblecolumns = "20"}
    local find_case = iup.toggle{title = "Case Sensitive"}
    local bt_find_next = iup.button{title = "Find Next", padding = "10x2"}
    local bt_find_close = iup.button{title = "Close", padding = "10x2"}

    function bt_find_next:action()
      local find_pos = tonumber(find_dlg.find_pos)
      local str_to_find = find_txt.value

      local casesensitive = (find_case.value == "ON")

      -- test again, because it can be called from the hot key
      if (not str_to_find or str_to_find:len()==0) then
        return
      end
```

```
  if (not find_pos) then
    find_pos = 1
  end

  local str = multitext.value

  local pos, end_pos = str_find(str, str_to_find, casesensitive, find_pos)
  if (not pos) then
    pos, end_pos = str_find(str, str_to_find, casesensitive, 1)  -- try again fr
  end

  if (pos) and (pos > 0) then
    pos = pos - 1
    find_dlg.find_pos = end_pos

    iup.SetFocus(multitext)
    multitext.selectionpos = pos..":"..end_pos

    local lin, col = iup.TextConvertPosToLinCol(multitext, pos)
    local pos = iup.TextConvertLinColToPos(multitext, lin, 0)  -- position at c
    multitext.scrolltopos = pos
  else
    find_dlg.find_pos = nil
    iup.Message("Warning", "Text not found.")
  end
end

function bt_find_close:action()
  iup.Hide(iup.GetDialog(self))  -- do not destroy, just hide
end

box = iup.vbox{
  iup.label{title = "Find What:"},
  find_txt,
  find_case,
  iup.hbox{
    iup.fill{},
    bt_find_next,
    bt_find_close,
    normalizesize="HORIZONTAL",
  },
```

```
      margin = "10x10",
      gap = "5",
    }

    find_dlg = iup.dialog{
      box,
      title = "Find",
      dialogframe = "Yes",
      defaultenter = bt_next,
      defaultesc = bt_close,
      parentdialog = iup.GetDialog(self)
      }

    -- Save the dialog to reuse it
    self.find_dialog = find_dlg -- from the main dialog */
  end

  -- centerparent first time, next time reuse the last position
  find_dlg:showxy(iup.CURRENT, iup.CURRENT)
end

function item_copy:action()
  local clipboard = iup.clipboard{text = multitext.selectedtext}
  clipboard:destroy()
end

function item_paste:action()
  local clipboard = iup.clipboard{}
  multitext.insert = clipboard.text
  clipboard:destroy()
  return iup.IGNORE -- avoid system processing for hot keys, to correctly parse li
end

function item_cut:action()
  local clipboard = iup.clipboard{text = multitext.selectedtext}
  multitext.selectedtext = ""
  clipboard:destroy()
end

function item_delete:action()
  multitext.selectedtext = ""
end
```

```
function item_select_all:action()
  iup.SetFocus(multitext)
  multitext.selection = "ALL"
end

function item_font:action()
  local font = multitext.font
  local fontdlg = iup.fontdlg{value = font, parentdialog=iup.GetDialog(self)}

  fontdlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(fontdlg.status) == 1) then
    multitext.font = fontdlg.value
    config:SetVariable("MainWindow", "Font", fontdlg.value)
  end

  fontdlg:destroy()
end

function item_toolbar:action()
  toggle_bar_visibility(self, toolbar_hb)
  config:SetVariable("MainWindow", "Toolbar", item_toolbar.value)
end

function item_statusbar:action()
  toggle_bar_visibility(self, lbl_statusbar)
  config:SetVariable("MainWindow", "Statusbar", item_statusbar.value)
end

function item_about:action()
  iup.Message("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣
end


--******************************** Main (Part 2/2) ****************************


recent_menu = iup.menu{}

file_menu = iup.menu{
  item_new,
```

```
  item_open,
  item_save,
  item_saveas,
  item_revert,
  iup.separator{},
  iup.submenu{title="Recent &Files", recent_menu},
  item_exit
  }

function file_menu:open_cb()
  if (multitext.dirty) then
    item_save.active = "YES"
  else
    item_save.active = "NO"
  end
  if (multitext.dirty and multitext.filename) then
    item_revert.active = "YES"
  else
    item_revert.active = "NO"
  end
end

edit_menu = iup.menu{
    item_cut,
    item_copy,
    item_paste,
    item_delete,
    iup.separator{},
    item_find,
    item_goto,
    iup.separator{},
    item_select_all
    }

function edit_menu:open_cb()
  local clipboard = iup.clipboard{}

  if (not clipboard.textavailable) then
    item_paste.active = "NO"
  else
    item_paste.active = "YES"
  end
```

```
  if (not multitext.selectedtext) then
    item_cut.active = "NO"
    item_delete.active = "NO"
    item_copy.active = "NO"
  else
    item_cut.active = "YES"
    item_delete.active = "YES"
    item_copy.active = "YES"
  end

  clipboard:destroy()
end

format_menu = iup.menu{item_font}
view_menu = iup.menu{item_toolbar, item_statusbar}
help_menu = iup.menu{item_about}

sub_menu_file = iup.submenu{file_menu, title = "&File"}
sub_menu_edit = iup.submenu{edit_menu, title = "&Edit"}
sub_menu_format = iup.submenu{format_menu, title = "F&ormat"}
sub_menu_view = iup.submenu{title = "&View", view_menu}
sub_menu_help = iup.submenu{help_menu, title = "&Help"}

menu = iup.menu{
  sub_menu_file,
  sub_menu_edit,
  sub_menu_format,
  sub_menu_view,
  sub_menu_help,
  }

btn_new = iup.button{image = "IUP_FileNew", flat = "Yes", action = item_new.action,
btn_open = iup.button{image = "IUP_FileOpen", flat = "Yes", action = item_open.actio
btn_save = iup.button{image = "IUP_FileSave", flat = "Yes", action = item_save.actio
btn_find = iup.button{image = "IUP_EditFind", flat = "Yes", action = item_find.actio
btn_cut = iup.button{image = "IUP_EditCut", flat = "Yes", action = item_cut.action,
btn_copy = iup.button{image =  "IUP_EditCopy", flat = "Yes", action = item_copy.act
btn_paste = iup.button{image = "IUP_EditPaste", flat = "Yes", action = item_paste.a

toolbar_hb = iup.hbox{
  btn_new,
```

```
  btn_open,
  btn_save,
  iup.label{separator="VERTICAL"},
  btn_cut,
  btn_copy,
  btn_paste,
  iup.label{separator="VERTICAL"},
  btn_find,
  margin = "5x5",
  gap = 2,
}

show_toolbar = config:GetVariableDef("MainWindow", "Toolbar", "ON")
if (show_toolbar == "OFF") then
  item_toolbar.value = "OFF"
  toolbar_hb.floating = "YES"
  toolbar_hb.visible = "NO"
end

vbox = iup.vbox{
  toolbar_hb,
  multitext,
  lbl_statusbar,
}

dlg = iup.dialog{
  vbox,
  title = "Simple Notepad",
  size = "HALFxHALF",
  menu = menu,
  close_cb = item_exit.action,
  multitext = multitext,
  dropfiles_cb = multitext.dropfiles_cb,
}

function dlg:k_any(c)
  if (c == iup.K_cN) then
    item_new:action()
  elseif (c == iup.K_cO) then
    item_open:action()
  elseif (c == iup.K_cS) then
    item_save:action()
```

```
  elseif (c == iup.K_cF) then
    item_find:action()
  elseif (c == iup.K_cG) then
    item_goto:action()
  end
end

config:RecentInit(recent_menu, 10)

-- parent for pre-defined dialogs in closed functions (IupMessage)
iup.SetGlobal("PARENTDIALOG", iup.SetHandleName(dlg))

config:DialogShow(dlg, "MainWindow")

-- initialize the current file
new_file(dlg)

-- open a file from the command line (allow file association in Windows)
if (arg and arg[1]) then
  filename = arg[1]
  open_file(dlg, filename)
end

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

3.11 外部帮助

我们可以为文本编辑器增加外部帮助的功能。*IUP* 提供了 *IupHelp* 函数来显示外部帮助。它会打开一个网页浏览器来显示指定的页面。在我们的例子中，我们在 *item_help_action_cb* 回调中调用 *IupHelp* 函数。我们用它显示了 **IUP** 的官方站点，它也可以用来显示本地的网页文件。在 Windows 下，这个函数甚至可以使用系统关联的浏览器打开任意文档。

C语言

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <iup.h>
#include <iup_config.h>
```

```
/********************************* Utilities ************************************

const char* str_filetitle(const char *filename)
{
  /* Start at the last character */
  int len = (int)strlen(filename);
  int offset = len - 1;
  while (offset != 0)
  {
    if (filename[offset] == '\\' || filename[offset] == '/')
    {
      offset++;
      break;
    }
    offset--;
  }
  return filename + offset;
}

int str_compare(const char *l, const char *r, int casesensitive)
{
  if (!l || !r)
    return 0;

  while (*l && *r)
  {
    int diff;
    char l_char = *l,
      r_char = *r;

    /* compute the difference of both characters */
    if (casesensitive)
      diff = l_char - r_char;
    else
      diff = tolower((int)l_char) - tolower((int)r_char);

    /* if they differ we have a result */
    if (diff != 0)
      return 0;
```

```
    /* otherwise process the next characters */
    ++l;
    ++r;
  }

  /* check also for terminator */
  if (*l == *r)
    return 1;

  if (*r == 0)
    return 1;  /* if second string is at terminator, then it is partially equal */

  return 0;
}

int str_find(const char *str, const char *str_to_find, int casesensitive)
{
  int i, str_len, str_to_find_len, count;

  if (!str || str[0] == 0 || !str_to_find || str_to_find[0] == 0)
    return -1;

  str_len = (int)strlen(str);
  str_to_find_len = (int)strlen(str_to_find);
  count = str_len - str_to_find_len;
  if (count < 0)
    return -1;

  count++;

  for (i = 0; i<count; i++)
  {
    if (str_compare(str, str_to_find, casesensitive))
      return i;

    str++;
  }

  return -1;
}

char* read_file(const char* filename)
```

```
{
  int size;
  char* str;
  FILE* file = fopen(filename, "rb");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return NULL;
  }

  /* calculate file size */
  fseek(file, 0, SEEK_END);
  size = ftell(file);
  fseek(file, 0, SEEK_SET);

  /* allocate memory for the file contents + nul terminator */
  str = malloc(size + 1);
  /* read all data at once */
  fread(str, size, 1, file);
  /* set the nul terminator */
  str[size] = 0;

  if (ferror(file))
    IupMessagef("Error", "Fail when reading from file: %s", filename);

  fclose(file);
  return str;
}

int write_file(const char* filename, const char* str, int count)
{
  FILE* file = fopen(filename, "w");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return 0;
  }

  fwrite(str, 1, count, file);

  if (ferror(file))
    IupMessagef("Error", "Fail when writing to file: %s", filename);
```

```
  fclose(file);
  return 1;
}

void new_file(Ihandle* ih)
{
  Ihandle* dlg = IupGetDialog(ih);
  Ihandle* multitext = IupGetDialogChild(dlg, "MULTITEXT");

  IupSetAttribute(dlg, "TITLE", "Untitled - Simple Notepad");
  IupSetAttribute(multitext, "FILENAME", NULL);
  IupSetAttribute(multitext, "DIRTY", "NO");
  IupSetAttribute(multitext, "VALUE", "");
}

void open_file(Ihandle* ih, const char* filename)
{
  char* str = read_file(filename);
  if (str)
  {
    Ihandle* dlg = IupGetDialog(ih);
    Ihandle* multitext = IupGetDialogChild(dlg, "MULTITEXT");
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

    IupSetfAttribute(dlg, "TITLE", "%s - Simple Notepad", str_filetitle(filename));
    IupSetStrAttribute(multitext, "FILENAME", filename);
    IupSetAttribute(multitext, "DIRTY", "NO");
    IupSetStrAttribute(multitext, "VALUE", str);

    IupConfigRecentUpdate(config, filename);

    free(str);
  }
}

void save_file(Ihandle* multitext)
{
  char* filename = IupGetAttribute(multitext, "FILENAME");
  char* str = IupGetAttribute(multitext, "VALUE");
  int count = IupGetInt(multitext, "COUNT");
  if (write_file(filename, str, count))
```

```
    IupSetAttribute(multitext, "DIRTY", "NO");
}

void saveas_file(Ihandle* multitext, const char* filename)
{
  char* str = IupGetAttribute(multitext, "VALUE");
  int count = IupGetInt(multitext, "COUNT");
  if (write_file(filename, str, count))
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

    IupSetfAttribute(IupGetDialog(multitext), "TITLE", "%s - Simple Notepad", str_
    IupSetStrAttribute(multitext, "FILENAME", filename);
    IupSetAttribute(multitext, "DIRTY", "NO");

    IupConfigRecentUpdate(config, filename);
  }
}

int save_check(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");
  if (IupGetInt(multitext, "DIRTY"))
  {
    switch (IupAlarm("Warning", "File not saved! Save it now?", "Yes", "No", "Cance
    {
    case 1:  /* save the changes and continue */
      save_file(multitext);
      break;
    case 2:  /* ignore the changes and continue */
      break;
    case 3:  /* cancel */
      return 0;
    }
  }
  return 1;
}

void toggle_bar_visibility(Ihandle* item, Ihandle* ih)
{
  if (IupGetInt(item, "VALUE"))
  {
```

```
    IupSetAttribute(ih, "FLOATING", "YES");
    IupSetAttribute(ih, "VISIBLE", "NO");
    IupSetAttribute(item, "VALUE", "OFF");
  }
  else
  {
    IupSetAttribute(ih, "FLOATING", "NO");
    IupSetAttribute(ih, "VISIBLE", "YES");
    IupSetAttribute(item, "VALUE", "ON");
  }

  IupRefresh(ih);  /* refresh the dialog layout */
}


/********************************** Callbacks ************************************


int dropfiles_cb(Ihandle* ih, const char* filename)
{
  if (save_check(ih))
    open_file(ih, filename);

  return IUP_DEFAULT;
}

int multitext_valuechanged_cb(Ihandle* multitext)
{
  IupSetAttribute(multitext, "DIRTY", "YES");
  return IUP_DEFAULT;
}

int file_menu_open_cb(Ihandle* ih)
{
  Ihandle* item_revert = IupGetDialogChild(ih, "ITEM_REVERT");
  Ihandle* item_save = IupGetDialogChild(ih, "ITEM_SAVE");
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");
  char* filename = IupGetAttribute(multitext, "FILENAME");
  int dirty = IupGetInt(multitext, "DIRTY");

  if (dirty)
    IupSetAttribute(item_save, "ACTIVE", "YES");
```

```
  else
    IupSetAttribute(item_save, "ACTIVE", "NO");

  if (dirty && filename)
    IupSetAttribute(item_revert, "ACTIVE", "YES");
  else
    IupSetAttribute(item_revert, "ACTIVE", "NO");
  return IUP_DEFAULT;
}

int edit_menu_open_cb(Ihandle* ih)
{
  Ihandle *clipboard = IupClipboard();

  Ihandle *item_paste = IupGetDialogChild(ih, "ITEM_PASTE");
  Ihandle *item_cut = IupGetDialogChild(ih, "ITEM_CUT");
  Ihandle *item_delete = IupGetDialogChild(ih, "ITEM_DELETE");
  Ihandle *item_copy = IupGetDialogChild(ih, "ITEM_COPY");
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  if (!IupGetInt(clipboard, "TEXTAVAILABLE"))
    IupSetAttribute(item_paste, "ACTIVE", "NO");
  else
    IupSetAttribute(item_paste, "ACTIVE", "YES");

  if (!IupGetAttribute(multitext, "SELECTEDTEXT"))
  {
    IupSetAttribute(item_cut, "ACTIVE", "NO");
    IupSetAttribute(item_delete, "ACTIVE", "NO");
    IupSetAttribute(item_copy, "ACTIVE", "NO");
  }
  else
  {
    IupSetAttribute(item_cut, "ACTIVE", "YES");
    IupSetAttribute(item_delete, "ACTIVE", "YES");
    IupSetAttribute(item_copy, "ACTIVE", "YES");
  }

  IupDestroy(clipboard);
  return IUP_DEFAULT;
}
```

```
int config_recent_cb(Ihandle* ih)
{
  if (save_check(ih))
  {
    char* filename = IupGetAttribute(ih, "TITLE");
    open_file(ih, filename);
  }
  return IUP_DEFAULT;
}

int multitext_caret_cb(Ihandle *ih, int lin, int col)
{
  Ihandle *lbl_statusbar = IupGetDialogChild(ih, "STATUSBAR");
  IupSetfAttribute(lbl_statusbar, "TITLE", "Lin %d, Col %d", lin, col);
  return IUP_DEFAULT;
}

int item_new_action_cb(Ihandle* item_new)
{
  if (save_check(item_new))
    new_file(item_new);

  return IUP_DEFAULT;
}

int item_open_action_cb(Ihandle* item_open)
{
  Ihandle *filedlg;

  if (!save_check(item_open))
    return IUP_DEFAULT;

  filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "OPEN");
  IupSetAttribute(filedlg, "EXTFILTER", "Text Files|*.txt|All Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_open));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    open_file(item_open, filename);
```

```
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_saveas_action_cb(Ihandle* item_saveas)
{
  Ihandle* multitext = IupGetDialogChild(item_saveas, "MULTITEXT");
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "SAVE");
  IupSetAttribute(filedlg, "EXTFILTER", "Text␣Files|*.txt|All␣Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_saveas));
  IupSetStrAttribute(filedlg, "FILE", IupGetAttribute(multitext, "FILENAME"));

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    saveas_file(multitext, filename);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_save_action_cb(Ihandle* item_save)
{
  Ihandle* multitext = IupGetDialogChild(item_save, "MULTITEXT");
  char* filename = IupGetAttribute(multitext, "FILENAME");
  if (!filename)
    item_saveas_action_cb(item_save);
  else
    save_file(multitext);
  return IUP_DEFAULT;
}

int item_revert_action_cb(Ihandle* item_revert)
{
  Ihandle* multitext = IupGetDialogChild(item_revert, "MULTITEXT");
  char* filename = IupGetAttribute(multitext, "FILENAME");
```

```
  open_file(item_revert, filename);
  return IUP_DEFAULT;
}

int item_exit_action_cb(Ihandle* item_exit)
{
  Ihandle* dlg = IupGetDialog(item_exit);
  Ihandle* config = (Ihandle*)IupGetAttribute(dlg, "CONFIG");

  if (!save_check(item_exit))
    return IUP_IGNORE;  /* to abort the CLOSE_CB callback */

  IupConfigDialogClosed(config, dlg, "MainWindow");
  IupConfigSave(config);
  IupDestroy(config);
  return IUP_CLOSE;
}

int goto_ok_action_cb(Ihandle* bt_ok)
{
  int line_count = IupGetInt(bt_ok, "TEXT_LINECOUNT");
  Ihandle* txt = IupGetDialogChild(bt_ok, "LINE_TEXT");
  int line = IupGetInt(txt, "VALUE");
  if (line < 1 || line >= line_count)
  {
    IupMessage("Error", "Invalid line number.");
    return IUP_DEFAULT;
  }

  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "1");
  return IUP_CLOSE;
}

int goto_cancel_action_cb(Ihandle* bt_ok)
{
  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "0");
  return IUP_CLOSE;
}

int item_goto_action_cb(Ihandle* item_goto)
{
  Ihandle* multitext = IupGetDialogChild(item_goto, "MULTITEXT");
```

```
  Ihandle *dlg, *box, *bt_ok, *bt_cancel, *txt, *lbl;
  int line_count = IupGetInt(multitext, "LINECOUNT");

  lbl = IupLabel(NULL);
  IupSetfAttribute(lbl, "TITLE", "Line Number [1-%d]:", line_count);
  txt = IupText(NULL);
  IupSetAttribute(txt, "MASK", IUP_MASK_UINT);  /* unsigned integer numbers only */
  IupSetAttribute(txt, "NAME", "LINE_TEXT");
  IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
  bt_ok = IupButton("OK", NULL);
  IupSetInt(bt_ok, "TEXT_LINECOUNT", line_count);
  IupSetAttribute(bt_ok, "PADDING", "10x2");
  IupSetCallback(bt_ok, "ACTION", (Icallback)goto_ok_action_cb);
  bt_cancel = IupButton("Cancel", NULL);
  IupSetCallback(bt_cancel, "ACTION", (Icallback)goto_cancel_action_cb);
  IupSetAttribute(bt_cancel, "PADDING", "10x2");

  box = IupVbox(
    lbl,
    txt,
    IupSetAttributes(IupHbox(
      IupFill(),
      bt_ok,
      bt_cancel,
      NULL), "NORMALIZESIZE=HORIZONTAL"),
    NULL);
  IupSetAttribute(box, "MARGIN", "10x10");
  IupSetAttribute(box, "GAP", "5");

  dlg = IupDialog(box);
  IupSetAttribute(dlg, "TITLE", "Go To Line");
  IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
  IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_ok);
  IupSetAttributeHandle(dlg, "DEFAULTESC", bt_cancel);
  IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_goto));

  IupPopup(dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(dlg, "STATUS") == 1)
  {
    int line = IupGetInt(txt, "VALUE");
    int pos;
```

```
    IupTextConvertLinColToPos(multitext, line, 0, &pos);
    IupSetInt(multitext, "CARETPOS", pos);
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }

  IupDestroy(dlg);

  return IUP_DEFAULT;
}

int find_next_action_cb(Ihandle* bt_next)
{
  Ihandle* multitext = (Ihandle*)IupGetAttribute(bt_next, "MULTITEXT");
  char* str = IupGetAttribute(multitext, "VALUE");
  int find_pos = IupGetInt(multitext, "FIND_POS");

  Ihandle* txt = IupGetDialogChild(bt_next, "FIND_TEXT");
  char* str_to_find = IupGetAttribute(txt, "VALUE");

  Ihandle* find_case = IupGetDialogChild(bt_next, "FIND_CASE");
  int casesensitive = IupGetInt(find_case, "VALUE");

  int pos = str_find(str + find_pos, str_to_find, casesensitive);
  if (pos >= 0)
    pos += find_pos;
  else if (find_pos > 0)
    pos = str_find(str, str_to_find, casesensitive);  /* try again from the start *

  if (pos >= 0)
  {
    int lin, col,
      end_pos = pos + (int)strlen(str_to_find);

    IupSetInt(multitext, "FIND_POS", end_pos);

    IupSetFocus(multitext);
    IupSetfAttribute(multitext, "SELECTIONPOS", "%d:%d", pos, end_pos);

    IupTextConvertPosToLinCol(multitext, pos, &lin, &col);
    IupTextConvertLinColToPos(multitext, lin, 0, &pos);  /* position at col=0, just
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }
```

```
  else
    IupMessage("Warning", "Text not found.");




  return IUP_DEFAULT;
}

int find_close_action_cb(Ihandle* bt_close)
{
  IupHide(IupGetDialog(bt_close));
  return IUP_DEFAULT;
}

int item_find_action_cb(Ihandle* item_find)
{
  Ihandle* dlg = (Ihandle*)IupGetAttribute(item_find, "FIND_DIALOG");
  if (!dlg)
  {
    Ihandle* multitext = IupGetDialogChild(item_find, "MULTITEXT");
    Ihandle *box, *bt_next, *bt_close, *txt, *find_case;


    txt = IupText(NULL);
    IupSetAttribute(txt, "NAME", "FIND_TEXT");
    IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
    find_case = IupToggle("Case Sensitive", NULL);
    IupSetAttribute(find_case, "NAME", "FIND_CASE");
    bt_next = IupButton("Find Next", NULL);
    IupSetAttribute(bt_next, "PADDING", "10x2");
    IupSetCallback(bt_next, "ACTION", (Icallback)find_next_action_cb);
    bt_close = IupButton("Close", NULL);
    IupSetCallback(bt_close, "ACTION", (Icallback)find_close_action_cb);
    IupSetAttribute(bt_close, "PADDING", "10x2");

    box = IupVbox(
      IupLabel("Find What:"),
      txt,
      find_case,
      IupSetAttributes(IupHbox(
        IupFill(),
```

```
        bt_next,
        bt_close,
        NULL), "NORMALIZESIZE=HORIZONTAL"),
      NULL);
    IupSetAttribute(box, "MARGIN", "10x10");
    IupSetAttribute(box, "GAP", "5");

    dlg = IupDialog(box);
    IupSetAttribute(dlg, "TITLE", "Find");
    IupSetAttribute(dlg, "DIALOGFRAME", "Yes");
    IupSetAttributeHandle(dlg, "DEFAULTENTER", bt_next);
    IupSetAttributeHandle(dlg, "DEFAULTESC", bt_close);
    IupSetAttributeHandle(dlg, "PARENTDIALOG", IupGetDialog(item_find));

    /* Save the multiline to acess it from the callbacks */
    IupSetAttribute(dlg, "MULTITEXT", (char*)multitext);

    /* Save the dialog to reuse it */
    IupSetAttribute(item_find, "FIND_DIALOG", (char*)dlg);
  }

  /* centerparent first time, next time reuse the last position */
  IupShowXY(dlg, IUP_CURRENT, IUP_CURRENT);

  return IUP_DEFAULT;
}

int item_copy_action_cb(Ihandle* item_copy)
{
  Ihandle* multitext = IupGetDialogChild(item_copy, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(clipboard, "TEXT", IupGetAttribute(multitext, "SELECTEDTEXT"));
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_paste_action_cb(Ihandle* item_paste)
{
  Ihandle* multitext = IupGetDialogChild(item_paste, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(multitext, "INSERT", IupGetAttribute(clipboard, "TEXT"));
  IupDestroy(clipboard);
```

```
  return IUP_DEFAULT;
}

int item_cut_action_cb(Ihandle* item_cut)
{
  Ihandle* multitext = IupGetDialogChild(item_cut, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(clipboard, "TEXT", IupGetAttribute(multitext, "SELECTEDTEXT"));
  IupSetAttribute(multitext, "SELECTEDTEXT", "");
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_delete_action_cb(Ihandle* item_delete)
{
  Ihandle* multitext = IupGetDialogChild(item_delete, "MULTITEXT");
  IupSetAttribute(multitext, "SELECTEDTEXT", "");
  return IUP_DEFAULT;
}

int item_select_all_action_cb(Ihandle* item_select_all)
{
  Ihandle* multitext = IupGetDialogChild(item_select_all, "MULTITEXT");
  IupSetFocus(multitext);
  IupSetAttribute(multitext, "SELECTION", "ALL");
  return IUP_DEFAULT;
}

int item_font_action_cb(Ihandle* item_font)
{
  Ihandle* multitext = IupGetDialogChild(item_font, "MULTITEXT");
  Ihandle* fontdlg = IupFontDlg();
  char* font = IupGetAttribute(multitext, "FONT");
  IupSetStrAttribute(fontdlg, "VALUE", font);
  IupSetAttributeHandle(fontdlg, "PARENTDIALOG", IupGetDialog(item_font));

  IupPopup(fontdlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(fontdlg, "STATUS") == 1)
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
    font = IupGetAttribute(fontdlg, "VALUE");
```

```
    IupSetStrAttribute(multitext, "FONT", font);

    IupConfigSetVariableStr(config, "MainWindow", "Font", font);
  }

  IupDestroy(fontdlg);
  return IUP_DEFAULT;
}

int item_toolbar_action_cb(Ihandle* item_toolbar)
{
  Ihandle* multitext = IupGetDialogChild(item_toolbar, "MULTITEXT");
  Ihandle* toolbar = IupGetChild(IupGetParent(multitext), 0);
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

  toggle_bar_visibility(item_toolbar, toolbar);

  IupConfigSetVariableStr(config, "MainWindow", "Toolbar", IupGetAttribute(item_too
  return IUP_DEFAULT;
}

int item_statusbar_action_cb(Ihandle* item_statusbar)
{
  Ihandle* multitext = IupGetDialogChild(item_statusbar, "MULTITEXT");
  Ihandle* statusbar = IupGetBrother(multitext);
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

  toggle_bar_visibility(item_statusbar, statusbar);

  IupConfigSetVariableStr(config, "MainWindow", "Statusbar", IupGetAttribute(item_s
  return IUP_DEFAULT;
}

int item_help_action_cb(void)
{
  IupHelp("http://www.tecgraf.puc-rio.br/iup");
  return IUP_DEFAULT;
}

int item_about_action_cb(void)
{
  IupMessage("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣S
```

```
  return IUP_DEFAULT;
}

/********************************** Main *****************************************/


int main(int argc, char **argv)
{
  Ihandle *dlg, *vbox, *multitext, *menu;
  Ihandle *sub_menu_file, *file_menu, *item_exit, *item_new, *item_open, *item_save
  Ihandle *sub_menu_edit, *edit_menu, *item_find, *item_goto, *item_copy, *item_pas
  Ihandle *btn_cut, *btn_copy, *btn_paste, *btn_find, *btn_new, *btn_open, *btn_sav
  Ihandle *sub_menu_format, *format_menu, *item_font;
  Ihandle *sub_menu_help, *help_menu, *item_help, *item_about;
  Ihandle *sub_menu_view, *view_menu, *item_toolbar, *item_statusbar;
  Ihandle *lbl_statusbar, *toolbar_hb, *recent_menu;
  Ihandle *config;
  const char* font;

  IupOpen(&argc, &argv);
  IupImageLibOpen();

  config = IupConfig();
  IupSetAttribute(config, "APP_NAME", "simple_notepad");
  IupConfigLoad(config);

  multitext = IupText(NULL);
    IupSetAttribute(multitext, "MULTILINE", "YES");
    IupSetAttribute(multitext, "EXPAND", "YES");
    IupSetAttribute(multitext, "NAME", "MULTITEXT");
    IupSetAttribute(multitext, "DIRTY", "NO");
    IupSetCallback(multitext, "CARET_CB", (Icallback)multitext_caret_cb);
    IupSetCallback(multitext, "VALUECHANGED_CB", (Icallback)multitext_valuechanged_
    IupSetCallback(multitext, "DROPFILES_CB", (Icallback)dropfiles_cb);

  font = IupConfigGetVariableStr(config, "MainWindow", "Font");
  if (font)
    IupSetStrAttribute(multitext, "FONT", font);

  lbl_statusbar = IupLabel("Lin 1, Col 1");
    IupSetAttribute(lbl_statusbar, "NAME", "STATUSBAR");
```

```
  IupSetAttribute(lbl_statusbar, "EXPAND", "HORIZONTAL");
  IupSetAttribute(lbl_statusbar, "PADDING", "10x5");

item_new = IupItem("New\tCtrl+N", NULL);
  IupSetAttribute(item_new, "IMAGE", "IUP_FileNew");
  IupSetCallback(item_new, "ACTION", (Icallback)item_new_action_cb);
btn_new = IupButton(NULL, NULL);
  IupSetAttribute(btn_new, "IMAGE", "IUP_FileNew");
  IupSetAttribute(btn_new, "FLAT", "Yes");
  IupSetCallback(btn_new, "ACTION", (Icallback)item_new_action_cb);
  IupSetAttribute(btn_new, "TIP", "New (Ctrl+N)");
  IupSetAttribute(btn_new, "CANFOCUS", "No");

item_open = IupItem("&Open...\tCtrl+O", NULL);
  IupSetAttribute(item_open, "IMAGE", "IUP_FileOpen");
  IupSetCallback(item_open, "ACTION", (Icallback)item_open_action_cb);
btn_open = IupButton(NULL, NULL);
  IupSetAttribute(btn_open, "IMAGE", "IUP_FileOpen");
  IupSetAttribute(btn_open, "FLAT", "Yes");
  IupSetCallback(btn_open, "ACTION", (Icallback)item_open_action_cb);
  IupSetAttribute(btn_open, "TIP", "Open (Ctrl+O)");
  IupSetAttribute(btn_open, "CANFOCUS", "No");

item_save = IupItem("Save\tCtrl+S", NULL);
  IupSetAttribute(item_save, "NAME", "ITEM_SAVE");
  IupSetAttribute(item_save, "IMAGE", "IUP_FileSave");
  IupSetCallback(item_save, "ACTION", (Icallback)item_save_action_cb);
btn_save = IupButton(NULL, NULL);
  IupSetAttribute(btn_save, "IMAGE", "IUP_FileSave");
  IupSetAttribute(btn_save, "FLAT", "Yes");
  IupSetCallback(btn_save, "ACTION", (Icallback)item_save_action_cb);
  IupSetAttribute(btn_save, "TIP", "Save (Ctrl+S)");
  IupSetAttribute(btn_save, "CANFOCUS", "No");

item_saveas = IupItem("Save &As...", NULL);
  IupSetAttribute(item_saveas, "NAME", "ITEM_SAVEAS");
  IupSetCallback(item_saveas, "ACTION", (Icallback)item_saveas_action_cb);

item_revert = IupItem("Revert", NULL);
  IupSetAttribute(item_revert, "NAME", "ITEM_REVERT");
  IupSetCallback(item_revert, "ACTION", (Icallback)item_revert_action_cb);
```

```
  item_exit = IupItem("E&xit", NULL);
    IupSetCallback(item_exit, "ACTION", (Icallback)item_exit_action_cb);

  item_find = IupItem("&Find...\tCtrl+F", NULL);
    IupSetAttribute(item_find, "IMAGE", "IUP_EditFind");
    IupSetCallback(item_find, "ACTION", (Icallback)item_find_action_cb);
  btn_find = IupButton(NULL, NULL);
    IupSetAttribute(btn_find, "IMAGE", "IUP_EditFind");
    IupSetAttribute(btn_find, "FLAT", "Yes");
    IupSetCallback(btn_find, "ACTION", (Icallback)item_find_action_cb);
    IupSetAttribute(btn_find, "TIP", "Find (Ctrl+F)");
    IupSetAttribute(btn_find, "CANFOCUS", "No");

  item_cut = IupItem("Cut\tCtrl+X", NULL);
    IupSetAttribute(item_cut, "NAME", "ITEM_CUT");
    IupSetAttribute(item_cut, "IMAGE", "IUP_EditCut");
    IupSetCallback(item_cut, "ACTION", (Icallback)item_cut_action_cb);
  item_copy = IupItem("Copy\tCtrl+C", NULL);
    IupSetAttribute(item_copy, "NAME", "ITEM_COPY");
    IupSetAttribute(item_copy, "IMAGE", "IUP_EditCopy");
    IupSetCallback(item_copy, "ACTION", (Icallback)item_copy_action_cb);
  item_paste = IupItem("Paste\tCtrl+V", NULL);
    IupSetAttribute(item_paste, "NAME", "ITEM_PASTE");
    IupSetAttribute(item_paste, "IMAGE", "IUP_EditPaste");
    IupSetCallback(item_paste, "ACTION", (Icallback)item_paste_action_cb);
  item_delete = IupItem("Delete\tDel", NULL);
    IupSetAttribute(item_delete, "IMAGE", "IUP_EditErase");
    IupSetAttribute(item_delete, "NAME", "ITEM_DELETE");
    IupSetCallback(item_delete, "ACTION", (Icallback)item_delete_action_cb);
  item_select_all = IupItem("Select All\tCtrl+A", NULL);
    IupSetCallback(item_select_all, "ACTION", (Icallback)item_select_all_action_cb)

  btn_cut = IupButton(NULL, NULL);
    IupSetAttribute(btn_cut, "IMAGE", "IUP_EditCut");
    IupSetAttribute(btn_cut, "FLAT", "Yes");
    IupSetCallback(btn_cut, "ACTION", (Icallback)item_cut_action_cb);
  btn_copy = IupButton(NULL, NULL);
    IupSetAttribute(btn_copy, "IMAGE", "IUP_EditCopy");
    IupSetAttribute(btn_copy, "FLAT", "Yes");
    IupSetCallback(btn_copy, "ACTION", (Icallback)item_copy_action_cb);
  btn_paste = IupButton(NULL, NULL);
    IupSetAttribute(btn_paste, "IMAGE", "IUP_EditPaste");
```

```
  IupSetAttribute(btn_paste, "FLAT", "Yes");
  IupSetCallback(btn_paste, "ACTION", (Icallback)item_paste_action_cb);

toolbar_hb = IupHbox(
  btn_new,
  btn_open,
  btn_save,
  IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
  btn_cut,
  btn_copy,
  btn_paste,
  IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
  btn_find,
  NULL);
IupSetAttribute(toolbar_hb, "MARGIN", "5x5");
IupSetAttribute(toolbar_hb, "GAP", "2");

item_toolbar = IupItem("&Toobar", NULL);
IupSetCallback(item_toolbar, "ACTION", (Icallback)item_toolbar_action_cb);
IupSetAttribute(item_toolbar, "VALUE", "ON");
item_statusbar = IupItem("&Statusbar", NULL);
IupSetCallback(item_statusbar, "ACTION", (Icallback)item_statusbar_action_cb);
IupSetAttribute(item_statusbar, "VALUE", "ON");

if (!IupConfigGetVariableIntDef(config, "MainWindow", "Toolbar", 1))
{
  IupSetAttribute(item_toolbar, "VALUE", "OFF");

  IupSetAttribute(toolbar_hb, "FLOATING", "YES");
  IupSetAttribute(toolbar_hb, "VISIBLE", "NO");
}

if (!IupConfigGetVariableIntDef(config, "MainWindow", "Statusbar", 1))
{
  IupSetAttribute(item_statusbar, "VALUE", "OFF");

  IupSetAttribute(lbl_statusbar, "FLOATING", "YES");
  IupSetAttribute(lbl_statusbar, "VISIBLE", "NO");
}

item_goto = IupItem("&Go␣To...\tCtrl+G", NULL);
  IupSetCallback(item_goto, "ACTION", (Icallback)item_goto_action_cb);
```

```
  item_font = IupItem("&Font...", NULL);
    IupSetCallback(item_font, "ACTION", (Icallback)item_font_action_cb);
  item_help = IupItem("&Help...", NULL);
    IupSetCallback(item_help, "ACTION", (Icallback)item_help_action_cb);
  item_about = IupItem("&About...", NULL);
    IupSetCallback(item_about, "ACTION", (Icallback)item_about_action_cb);

  recent_menu = IupMenu(NULL);

  file_menu = IupMenu(
    item_new,
    item_open,
    item_save,
    item_saveas,
    item_revert,
    IupSeparator(),
    IupSubmenu("Recent &Files", recent_menu),
    item_exit,
    NULL);
  edit_menu = IupMenu(
    item_cut,
    item_copy,
    item_paste,
    item_delete,
    IupSeparator(),
    item_find,
    item_goto,
    IupSeparator(),
    item_select_all,
    NULL);
  format_menu = IupMenu(
    item_font,
    NULL);
  view_menu = IupMenu(
    item_toolbar,
    item_statusbar,
    NULL);
  help_menu = IupMenu(
    item_help,
    item_about,
    NULL);
```

```
IupSetCallback(file_menu, "OPEN_CB", (Icallback)file_menu_open_cb);
IupSetCallback(edit_menu, "OPEN_CB", (Icallback)edit_menu_open_cb);

sub_menu_file = IupSubmenu("&File", file_menu);
sub_menu_edit = IupSubmenu("&Edit", edit_menu);
sub_menu_format = IupSubmenu("F&ormat", format_menu);
sub_menu_view = IupSubmenu("&View", view_menu);
sub_menu_help = IupSubmenu("&Help", help_menu);

menu = IupMenu(
  sub_menu_file,
  sub_menu_edit,
  sub_menu_format,
  sub_menu_view,
  sub_menu_help,
  NULL);

vbox = IupVbox(
  toolbar_hb,
  multitext,
  lbl_statusbar,
  NULL);

dlg = IupDialog(vbox);
IupSetAttributeHandle(dlg, "MENU", menu);
IupSetAttribute(dlg, "SIZE", "HALFxHALF");
IupSetCallback(dlg, "CLOSE_CB", (Icallback)item_exit_action_cb);
IupSetCallback(dlg, "DROPFILES_CB", (Icallback)dropfiles_cb);

IupSetAttribute(dlg, "CONFIG", (char*)config);

/* parent for pre-defined dialogs in closed functions (IupMessage) */
IupSetAttributeHandle(NULL, "PARENTDIALOG", dlg);

IupSetCallback(dlg, "K_cN", (Icallback)item_new_action_cb);
IupSetCallback(dlg, "K_cO", (Icallback)item_open_action_cb);
IupSetCallback(dlg, "K_cS", (Icallback)item_save_action_cb);
IupSetCallback(dlg, "K_cF", (Icallback)item_find_action_cb);
IupSetCallback(dlg, "K_cG", (Icallback)item_goto_action_cb);

IupConfigRecentInit(config, recent_menu, config_recent_cb, 10);
```

```
  IupConfigDialogShow(config, dlg, "MainWindow");

  /* initialize the current file */
  new_file(dlg);

  /* open a file from the command line (allow file association in Windows) */
  if (argc > 1 && argv[1])
  {
    const char* filename = argv[1];
    open_file(dlg, filename);
  }

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")
require("iupluaimglib")


--********************************** Utilities *********************************


function str_find(str, str_to_find, casesensitive, start)
  if (not casesensitive) then
    return str_find(string.lower(str), string.lower(str_to_find), true, start)
  end

  return string.find(str, str_to_find, start, true)
end

function str_filetitle(filename)
  local filename = string.gsub(filename, "\\", "/")
  filename = string.reverse(filename)
  final = string.find(filename, '/')
  filename = string.sub(filename, 1, final-1)
  filename = string.reverse(filename)
  return filename
```

```
end

function read_file(filename)
  local ifile = io.open(filename, "r")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return nil
  end

  local str = ifile:read("*a")
  if (not str) then
    iup.Message("Error", "Fail when reading from file: " .. filename)
    return nil
  end

  ifile:close()
  return str
end

function write_file(filename, str)
  local ifile = io.open(filename, "w")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return false
  end

  if (not ifile:write(str)) then
    iup.Message("Error", "Fail when writing to file: " .. filename)
  end

  ifile:close()
  return true
end

function new_file(ih)
  local dlg = iup.GetDialog(ih)
  local multitext = dlg.multitext

  dlg.title = "Untitled - Simple Notepad"
  multitext.filename = nil
  multitext.dirty = nil
  multitext.value = ""
```

```
end

function open_file(ih, filename)
  local str = read_file(filename)
  if (str) then
    local dlg = iup.GetDialog(ih)
    local multitext = dlg.multitext
    local config = multitext.config

    dlg.title = str_filetitle(filename)..” – Simple Notepad”
    multitext.filename = filename
    multitext.dirty = nil
    multitext.value = str

    config:RecentUpdate(filename)
  end
end

function save_file(multitext)
  if (write_file(multitext.filename, multitext.value)) then
    multitext.dirty = nil
  end
end

function saveas_file(multitext, filename)
  if (write_file(filename, multitext.value)) then
    local dlg = iup.GetDialog(multitext)
    local config = multitext.config

    dlg.title = str_filetitle(filename)..” – Simple Notepad”
    multitext.filename = filename
    multitext.dirty = nil

    config:RecentUpdate(filename)
  end
end

function save_check(ih)
  local dlg = iup.GetDialog(ih)
  local multitext = dlg.multitext

  if (multitext.dirty) then
```

```
    local resp = iup.Alarm("Warning", "File not saved! Save it now?", "Yes", "No",
    if resp == 1 then -- save the changes and continue
      save_file(multitext)
    elseif resp == 3 then  -- cancel
      return false
    else  -- ignore the changes and continue
    end
  end
  return true
end

function toggle_bar_visibility(item, bar)
  if (item.value == "ON") then
    bar.floating = "YES"
    bar.visible = "NO"
    item.value = "OFF"
  else
    bar.floating = "NO"
    bar.visible = "YES"
    item.value = "ON"
  end
  iup.Refresh(bar)  -- refresh the dialog layout
end


--******************************** Main (Part 1/2) ****************************


config = iup.config{}
config.app_name = "simple_notepad"
config:Load()

lbl_statusbar = iup.label{title = "Lin 1, Col 1", expand = "HORIZONTAL", padding =

multitext = iup.text{
  multiline = "YES",
  expand = "YES",
  config = config,
  dirty = nil,
}

font = config:GetVariable("MainWindow", "Font")
```

```
if (font) then
  multitext.font = font
end

item_new = iup.item{title = "&New...\tCtrl+N", image = "IUP_FileNew"}
item_open = iup.item{title = "&Open...\tCtrl+O", image = "IUP_FileOpen"}
item_save = iup.item{title="&Save\tCtrl+S"}
item_saveas = iup.item{title="Save &As...", image = "IUP_FileSave"}
item_font = iup.item{title="&Font..."}
item_about = iup.item{title="&About..."}
item_find = iup.item{title="&Find...\tCtrl+F", image = "IUP_EditFind"}
item_goto = iup.item{title="&Go To..."}
item_copy = iup.item{title="&Copy\tCtrl+C", image = "IUP_EditCopy"}
item_paste = iup.item{title="&Paste\tCtrl+V", image = "IUP_EditPaste"}
item_cut = iup.item{title="Cu&t\tCtrl+X", image = "IUP_EditCut"}
item_delete = iup.item{title="&Delete\tDel", image = "IUP_EditErase"}
item_select_all = iup.item{title="Select &All\tCtrl+A"}
item_revert = iup.item{title="&Revert"}
item_exit = iup.item{title="E&xit"}
item_toolbar = iup.item{title="&Toobar", value="ON"}
item_statusbar = iup.item{title="&Statusbar", value="ON"}
item_help = iup.item{title="&Help..."}

show_statusbar = config:GetVariableDef("MainWindow", "Statusbar", "ON")
if (show_statusbar == "OFF") then
  item_statusbar.value = "OFF"
  lbl_statusbar.floating = "YES"
  lbl_statusbar.visible = "NO"
end


--********************************** Callbacks **********************************


function multitext:dropfiles_cb(filename)
  if (save_check(self)) then
    open_file(self, filename)
  end
end

function multitext:valuechanged_cb()
  self.dirty = "YES"
```

```
end

function config:recent_cb()
  if (save_check(self)) then
    local filename = self.title
    open_file(self, filename)
  end
end

function multitext:caret_cb(lin, col)
  lbl_statusbar.title = "Lin "..lin.." , Col "..col
end

function item_new:action()
  if save_check(self) then
    new_file(self)
  end
end

function item_open:action()
  if not save_check(self) then
    return
  end

  local filedlg = iup.filedlg{
    dialogtype = "OPEN",
    filter = "*.txt",
    filterinfo = "Text Files",
    parentdialog=iup.GetDialog(self)
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    open_file(self, filename)
  end

  filedlg:destroy()
end

function item_saveas:action()
```

```
  local filedlg = iup.filedlg{
    dialogtype = "SAVE",
    filter = "*.txt",
    filterinfo = "Text Files",
    parentdialog = iup.GetDialog(self),
    file = multitext.filename,
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    saveas_file(multitext, filename)
  end

  filedlg:destroy()
end

function item_save:action()
  if (not multitext.filename) then
    item_saveas:action()
  else
    save_file(multitext)
  end
end

function item_revert:action()
  open_file(self, multitext.filename)
end

function item_exit:action()
  if not save_check(self) then
    return iup.IGNORE  -- to abort the CLOSE_CB callback
  end

  config:DialogClosed(iup.GetDialog(self), "MainWindow")
  config:Save()
  config:destroy()
  return iup.CLOSE
end

function item_goto:action()
```

```
local line_count = multitext.linecount
local lbl_goto = iup.label{title = "Line Number [1-"..line_count.."]:"}
local txt_goto = iup.text{mask = iup.MASK_UINT, visiblecolumns = 20} --unsigned i

local bt_goto_ok = iup.button{title = "OK", text_linecount = 0, padding = "10x2"}
bt_goto_ok.text_linecount = line_count
function bt_goto_ok:action()
  local line_count = tonumber(self.text_linecount)
  local line = tonumber(txt_goto.value)
  if (line < 1 or line >= line_count) then
    iup.Message("Error", "Invalid line number.")
    return
  end
  goto_dlg.status = 1
  return iup.CLOSE
end

local bt_goto_cancel = iup.button{title = "Cancel", padding = "10x2"}
function bt_goto_cancel:action()
  goto_dlg.status = 0
  return iup.CLOSE
end

local box = iup.vbox{
  lbl_goto,
  txt_goto,
  iup.hbox{
    iup.fill{},
    bt_goto_ok,
    bt_goto_cancel,
    normalizesize="HORIZONTAL",
  },
  margin = "10x10",
  gap = "5",
}
goto_dlg = iup.dialog{
  box,
  title = "Go To Line",
  dialogframe = "Yes",
  defaultenter = bt_goto_ok,
  defaultesc = bt_goto_cancel,
  parentdialog = iup.GetDialog(self)
```

```
  }

  goto_dlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(goto_dlg.status) == 1) then
    local line = txt_goto.value
    local pos = iup.TextConvertLinColToPos(multitext, line, 0)
    multitext.caretpos = pos
    multitext.scrolltopos = pos
  end

  goto_dlg:destroy()
end

function item_find:action()
  local find_dlg = self.find_dialog
  if (not find_dlg) then
    local find_txt = iup.text{visiblecolumns = "20"}
    local find_case = iup.toggle{title = "Case Sensitive"}
    local bt_find_next = iup.button{title = "Find Next", padding = "10x2"}
    local bt_find_close = iup.button{title = "Close", padding = "10x2"}

    function bt_find_next:action()
      local find_pos = tonumber(find_dlg.find_pos)
      local str_to_find = find_txt.value

      local casesensitive = (find_case.value == "ON")

      -- test again, because it can be called from the hot key
      if (not str_to_find or str_to_find:len()==0) then
        return
      end

      if (not find_pos) then
        find_pos = 1
      end

      local str = multitext.value

      local pos, end_pos = str_find(str, str_to_find, casesensitive, find_pos)
      if (not pos) then
        pos, end_pos = str_find(str, str_to_find, casesensitive, 1)  -- try again fr
```

```
  end

  if (pos) and (pos > 0) then
    pos = pos - 1
    find_dlg.find_pos = end_pos

    iup.SetFocus(multitext)
    multitext.selectionpos = pos..":"..end_pos

    local lin, col = iup.TextConvertPosToLinCol(multitext, pos)
    local pos = iup.TextConvertLinColToPos(multitext, lin, 0) -- position at c
    multitext.scrolltopos = pos
  else
    find_dlg.find_pos = nil
    iup.Message("Warning", "Text not found.")
  end
end

function bt_find_close:action()
  iup.Hide(iup.GetDialog(self)) -- do not destroy, just hide
end

box = iup.vbox{
  iup.label{title = "Find What:"},
  find_txt,
  find_case,
  iup.hbox{
    iup.fill{},
    bt_find_next,
    bt_find_close,
    normalizesize="HORIZONTAL",
  },
  margin = "10x10",
  gap = "5",
}

find_dlg = iup.dialog{
  box,
  title = "Find",
  dialogframe = "Yes",
  defaultenter = bt_next,
  defaultesc = bt_close,
```

```
      parentdialog = iup.GetDialog(self)
      }

    -- Save the dialog to reuse it
    self.find_dialog = find_dlg -- from the main dialog */
  end

  -- centerparent first time, next time reuse the last position
  find_dlg:showxy(iup.CURRENT, iup.CURRENT)
end

function item_copy:action()
  local clipboard = iup.clipboard{text = multitext.selectedtext}
  clipboard:destroy()
end

function item_paste:action()
  local clipboard = iup.clipboard{}
  multitext.insert = clipboard.text
  clipboard:destroy()
  return iup.IGNORE -- avoid system processing for hot keys, to correctly parse li
end

function item_cut:action()
  local clipboard = iup.clipboard{text = multitext.selectedtext}
  multitext.selectedtext = ""
  clipboard:destroy()
end

function item_delete:action()
  multitext.selectedtext = ""
end

function item_select_all:action()
  iup.SetFocus(multitext)
  multitext.selection = "ALL"
end

function item_font:action()
  local font = multitext.font
  local fontdlg = iup.fontdlg{value = font, parentdialog=iup.GetDialog(self)}
```

```
  fontdlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(fontdlg.status) == 1) then
    multitext.font = fontdlg.value
    config:SetVariable("MainWindow", "Font", fontdlg.value)
  end

  fontdlg:destroy()
end

function item_toolbar:action()
  toggle_bar_visibility(self, toolbar_hb)
  config:SetVariable("MainWindow", "Toolbar", item_toolbar.value)
end

function item_statusbar:action()
  toggle_bar_visibility(self, lbl_statusbar)
  config:SetVariable("MainWindow", "Statusbar", item_statusbar.value)
end

function item_help:action()
  iup.Help("http://www.tecgraf.puc-rio.br/iup")
end

function item_about:action()
  iup.Message("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣
end


--********************************* Main (Part 2/2) ****************************


recent_menu = iup.menu{}

file_menu = iup.menu{
  item_new,
  item_open,
  item_save,
  item_saveas,
  item_revert,
  iup.separator{},
  iup.submenu{title="Recent␣&Files", recent_menu},
```

```
  item_exit
  }

function file_menu:open_cb()
  if (multitext.dirty) then
    item_save.active = "YES"
  else
    item_save.active = "NO"
  end
  if (multitext.dirty and multitext.filename) then
    item_revert.active = "YES"
  else
    item_revert.active = "NO"
  end
end

edit_menu = iup.menu{
    item_cut,
    item_copy,
    item_paste,
    item_delete,
    iup.separator{},
    item_find,
    item_goto,
    iup.separator{},
    item_select_all
    }

function edit_menu:open_cb()
  local clipboard = iup.clipboard{}

  if (not clipboard.textavailable) then
    item_paste.active = "NO"
  else
    item_paste.active = "YES"
  end

  if (not multitext.selectedtext) then
    item_cut.active = "NO"
    item_delete.active = "NO"
    item_copy.active = "NO"
  else
```

```
    item_cut.active = "YES"
    item_delete.active = "YES"
    item_copy.active = "YES"
  end

  clipboard:destroy()
end

format_menu = iup.menu{item_font}
view_menu = iup.menu{item_toolbar, item_statusbar}
help_menu = iup.menu{item_help, item_about}

sub_menu_file = iup.submenu{file_menu, title = "&File"}
sub_menu_edit = iup.submenu{edit_menu, title = "&Edit"}
sub_menu_format = iup.submenu{format_menu, title = "F&ormat"}
sub_menu_view = iup.submenu{title = "&View", view_menu}
sub_menu_help = iup.submenu{help_menu, title = "&Help"}

menu = iup.menu{
  sub_menu_file,
  sub_menu_edit,
  sub_menu_format,
  sub_menu_view,
  sub_menu_help,
  }

btn_new = iup.button{image = "IUP_FileNew", flat = "Yes", action = item_new.action,
btn_open = iup.button{image = "IUP_FileOpen", flat = "Yes", action = item_open.actio
btn_save = iup.button{image = "IUP_FileSave", flat = "Yes", action = item_save.actio
btn_find = iup.button{image = "IUP_EditFind", flat = "Yes", action = item_find.actio
btn_cut = iup.button{image = "IUP_EditCut", flat = "Yes", action = item_cut.action,
btn_copy = iup.button{image =  "IUP_EditCopy", flat = "Yes", action = item_copy.act
btn_paste = iup.button{image = "IUP_EditPaste", flat = "Yes", action = item_paste.a

toolbar_hb = iup.hbox{
  btn_new,
  btn_open,
  btn_save,
  iup.label{separator="VERTICAL"},
  btn_cut,
  btn_copy,
  btn_paste,
```

```
  iup.label{separator="VERTICAL"},
  btn_find,
  margin = "5x5",
  gap = 2,
}

show_toolbar = config:GetVariableDef("MainWindow", "Toolbar", "ON")
if (show_toolbar == "OFF") then
  item_toolbar.value = "OFF"
  toolbar_hb.floating = "YES"
  toolbar_hb.visible = "NO"
end

vbox = iup.vbox{
  toolbar_hb,
  multitext,
  lbl_statusbar,
}

dlg = iup.dialog{
  vbox,
  title = "Simple Notepad",
  size = "HALFxHALF",
  menu = menu,
  close_cb = item_exit.action,
  multitext = multitext,
  dropfiles_cb = multitext.dropfiles_cb,
}

function dlg:k_any(c)
  if (c == iup.K_cN) then
    item_new:action()
  elseif (c == iup.K_cO) then
    item_open:action()
  elseif (c == iup.K_cS) then
    item_save:action()
  elseif (c == iup.K_cF) then
    item_find:action()
  elseif (c == iup.K_cG) then
    item_goto:action()
  end
end
```

```
config:RecentInit(recent_menu, 10)

-- parent for pre-defined dialogs in closed functions (IupMessage)
iup.SetGlobal("PARENTDIALOG", iup.SetHandleName(dlg))

config:DialogShow(dlg, "MainWindow")

-- initialize the current file
new_file(dlg)

-- open a file from the command line (allow file association in Windows)
if (arg and arg[1]) then
  filename = arg[1]
  open_file(dlg, filename)
end

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

3.12 最后的讨论

最后第三章马上就要结束了，当你进行到这里，说明你已经可以使用 **IUP** 创建一个功能全面的记事本程序。

在本章，我们从 30 行的代码开始一直到最后的 1100 行[8]，构建了一个带有文件读写，剪贴板访问，文本搜索功能的记事本。

最后，我们给它添加两个功能：使用热键进行替换和查找。之后我们重新组织了代码，并加上了一些注释。

对于这个程序而言，还有许多可以改进的地方，欢迎大家贡献代码。

C语言

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <iup.h>
#include <iup_config.h>
```

[8]Lua 代码。

```
/********************************* Utilities ***********************************

const char* str_filetitle(const char *filename)
{
  /* Start at the last character */
  int len = (int)strlen(filename);
  int offset = len - 1;
  while (offset != 0)
  {
    if (filename[offset] == '\\' || filename[offset] == '/')
    {
      offset++;
      break;
    }
    offset--;
  }
  return filename + offset;
}

int str_compare(const char *l, const char *r, int casesensitive)
{
  if (!l || !r)
    return 0;

  while (*l && *r)
  {
    int diff;
    char l_char = *l,
      r_char = *r;

    /* compute the difference of both characters */
    if (casesensitive)
      diff = l_char - r_char;
    else
      diff = tolower((int)l_char) - tolower((int)r_char);

    /* if they differ we have a result */
    if (diff != 0)
      return 0;

    /* otherwise process the next characters */
```

```
    ++l;
    ++r;
  }

  /* check also for terminator */
  if (*l == *r)
    return 1;

  if (*r == 0)
    return 1;  /* if second string is at terminator, then it is partially equal */

  return 0;
}

int str_find(const char *str, const char *str_to_find, int casesensitive)
{
  int i, str_len, str_to_find_len, count;

  if (!str || str[0] == 0 || !str_to_find || str_to_find[0] == 0)
    return -1;

  str_len = (int)strlen(str);
  str_to_find_len = (int)strlen(str_to_find);
  count = str_len - str_to_find_len;
  if (count < 0)
    return -1;

  count++;

  for (i = 0; i<count; i++)
  {
    if (str_compare(str, str_to_find, casesensitive))
      return i;

    str++;
  }

  return -1;
}

char* read_file(const char* filename)
{
```

```
  int size;
  char* str;
  FILE* file = fopen(filename, "rb");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return NULL;
  }

  /* calculate file size */
  fseek(file, 0, SEEK_END);
  size = ftell(file);
  fseek(file, 0, SEEK_SET);

  /* allocate memory for the file contents + nul terminator */
  str = malloc(size + 1);
  /* read all data at once */
  fread(str, size, 1, file);
  /* set the nul terminator */
  str[size] = 0;

  if (ferror(file))
    IupMessagef("Error", "Fail when reading from file: %s", filename);

  fclose(file);
  return str;
}

int write_file(const char* filename, const char* str, int count)
{
  FILE* file = fopen(filename, "w");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return 0;
  }

  fwrite(str, 1, count, file);

  if (ferror(file))
    IupMessagef("Error", "Fail when writing to file: %s", filename);
```

```
  fclose(file);
  return 1;
}

void new_file(Ihandle* ih)
{
  Ihandle* dlg = IupGetDialog(ih);
  Ihandle* multitext = IupGetDialogChild(dlg, "MULTITEXT");

  IupSetAttribute(dlg, "TITLE", "Untitled - Simple Notepad");
  IupSetAttribute(multitext, "FILENAME", NULL);
  IupSetAttribute(multitext, "DIRTY", "NO");
  IupSetAttribute(multitext, "VALUE", "");
}

void open_file(Ihandle* ih, const char* filename)
{
  char* str = read_file(filename);
  if (str)
  {
    Ihandle* dlg = IupGetDialog(ih);
    Ihandle* multitext = IupGetDialogChild(dlg, "MULTITEXT");
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

    IupSetfAttribute(dlg, "TITLE", "%s - Simple Notepad", str_filetitle(filename));
    IupSetStrAttribute(multitext, "FILENAME", filename);
    IupSetAttribute(multitext, "DIRTY", "NO");
    IupSetStrAttribute(multitext, "VALUE", str);

    IupConfigRecentUpdate(config, filename);

    free(str);
  }
}

void save_file(Ihandle* multitext)
{
  char* filename = IupGetAttribute(multitext, "FILENAME");
  char* str = IupGetAttribute(multitext, "VALUE");
  int count = IupGetInt(multitext, "COUNT");
  if (write_file(filename, str, count))
    IupSetAttribute(multitext, "DIRTY", "NO");
```

```
}

void saveas_file(Ihandle* multitext, const char* filename)
{
  char* str = IupGetAttribute(multitext, "VALUE");
  int count = IupGetInt(multitext, "COUNT");
  if (write_file(filename, str, count))
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

    IupSetfAttribute(IupGetDialog(multitext), "TITLE", "%s - Simple Notepad", str_
    IupSetStrAttribute(multitext, "FILENAME", filename);
    IupSetAttribute(multitext, "DIRTY", "NO");

    IupConfigRecentUpdate(config, filename);
  }
}

int save_check(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");
  if (IupGetInt(multitext, "DIRTY"))
  {
    switch (IupAlarm("Warning", "File not saved! Save it now?", "Yes", "No", "Cance
    {
    case 1:  /* save the changes and continue */
      save_file(multitext);
      break;
    case 2:  /* ignore the changes and continue */
      break;
    case 3:  /* cancel */
      return 0;
    }
  }
  return 1;
}

void toggle_bar_visibility(Ihandle* item, Ihandle* ih)
{
  if (IupGetInt(item, "VALUE"))
  {
    IupSetAttribute(ih, "FLOATING", "YES");
```

```
    IupSetAttribute(ih, "VISIBLE", "NO");
    IupSetAttribute(item, "VALUE", "OFF");
  }
  else
  {
    IupSetAttribute(ih, "FLOATING", "NO");
    IupSetAttribute(ih, "VISIBLE", "YES");
    IupSetAttribute(item, "VALUE", "ON");
  }

  IupRefresh(ih);  /* refresh the dialog layout */
}

void set_find_replace_visibility(Ihandle* find_dlg, int show_replace)
{
  Ihandle* replace_txt = IupGetDialogChild(find_dlg, "REPLACE_TEXT");
  Ihandle* replace_lbl = IupGetDialogChild(find_dlg, "REPLACE_LABEL");
  Ihandle* replace_bt = IupGetDialogChild(find_dlg, "REPLACE_BUTTON");

  if (show_replace)
  {
    IupSetAttribute(replace_txt, "VISIBLE", "Yes");
    IupSetAttribute(replace_lbl, "VISIBLE", "Yes");
    IupSetAttribute(replace_bt, "VISIBLE", "Yes");
    IupSetAttribute(replace_txt, "FLOATING", "No");
    IupSetAttribute(replace_lbl, "FLOATING", "No");
    IupSetAttribute(replace_bt, "FLOATING", "No");

    IupSetAttribute(find_dlg, "TITLE", "Replace");
  }
  else
  {
    IupSetAttribute(replace_txt, "FLOATING", "Yes");
    IupSetAttribute(replace_lbl, "FLOATING", "Yes");
    IupSetAttribute(replace_bt, "FLOATING", "Yes");
    IupSetAttribute(replace_txt, "VISIBLE", "No");
    IupSetAttribute(replace_lbl, "VISIBLE", "No");
    IupSetAttribute(replace_bt, "VISIBLE", "No");

    IupSetAttribute(find_dlg, "TITLE", "Find");
  }
```

```
  IupSetAttribute(find_dlg, "SIZE", NULL);  /* force a dialog resize on the IupRefr
  IupRefresh(find_dlg);
}


/********************************* Callbacks ************************************


int dropfiles_cb(Ihandle* ih, const char* filename)
{
  if (save_check(ih))
    open_file(ih, filename);

  return IUP_DEFAULT;
}

int multitext_valuechanged_cb(Ihandle* multitext)
{
  IupSetAttribute(multitext, "DIRTY", "YES");
  return IUP_DEFAULT;
}

int file_menu_open_cb(Ihandle* ih)
{
  Ihandle* item_revert = IupGetDialogChild(ih, "ITEM_REVERT");
  Ihandle* item_save = IupGetDialogChild(ih, "ITEM_SAVE");
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");
  char* filename = IupGetAttribute(multitext, "FILENAME");
  int dirty = IupGetInt(multitext, "DIRTY");

  if (dirty)
    IupSetAttribute(item_save, "ACTIVE", "YES");
  else
    IupSetAttribute(item_save, "ACTIVE", "NO");

  if (dirty && filename)
    IupSetAttribute(item_revert, "ACTIVE", "YES");
  else
    IupSetAttribute(item_revert, "ACTIVE", "NO");
  return IUP_DEFAULT;
}
```

```
int edit_menu_open_cb(Ihandle* ih)
{
  Ihandle *clipboard = IupClipboard();
  Ihandle* find_dlg = (Ihandle*)IupGetAttribute(ih, "FIND_DIALOG");

  Ihandle *item_paste = IupGetDialogChild(ih, "ITEM_PASTE");
  Ihandle *item_cut = IupGetDialogChild(ih, "ITEM_CUT");
  Ihandle *item_delete = IupGetDialogChild(ih, "ITEM_DELETE");
  Ihandle *item_copy = IupGetDialogChild(ih, "ITEM_COPY");
  Ihandle *item_find_next = IupGetDialogChild(ih, "ITEM_FINDNEXT");
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  if (!IupGetInt(clipboard, "TEXTAVAILABLE"))
    IupSetAttribute(item_paste, "ACTIVE", "NO");
  else
    IupSetAttribute(item_paste, "ACTIVE", "YES");

  if (!IupGetAttribute(multitext, "SELECTEDTEXT"))
  {
    IupSetAttribute(item_cut, "ACTIVE", "NO");
    IupSetAttribute(item_delete, "ACTIVE", "NO");
    IupSetAttribute(item_copy, "ACTIVE", "NO");
  }
  else
  {
    IupSetAttribute(item_cut, "ACTIVE", "YES");
    IupSetAttribute(item_delete, "ACTIVE", "YES");
    IupSetAttribute(item_copy, "ACTIVE", "YES");
  }

  if (find_dlg)
  {
    Ihandle* txt = IupGetDialogChild(find_dlg, "FIND_TEXT");
    char* str_to_find = IupGetAttribute(txt, "VALUE");

    if (!str_to_find || str_to_find[0] == 0)
      IupSetAttribute(item_find_next, "ACTIVE", "NO");
    else
      IupSetAttribute(item_find_next, "ACTIVE", "Yes");
  }
  else
    IupSetAttribute(item_find_next, "ACTIVE", "NO");
```

```
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int config_recent_cb(Ihandle* ih)
{
  if (save_check(ih))
  {
    char* filename = IupGetAttribute(ih, "TITLE");
    open_file(ih, filename);
  }
  return IUP_DEFAULT;
}

int multitext_caret_cb(Ihandle *ih, int lin, int col)
{
  Ihandle *lbl_statusbar = IupGetDialogChild(ih, "STATUSBAR");
  IupSetfAttribute(lbl_statusbar, "TITLE", "Lin %d, Col %d", lin, col);
  return IUP_DEFAULT;
}

int item_new_action_cb(Ihandle* item_new)
{
  if (save_check(item_new))
    new_file(item_new);

  return IUP_DEFAULT;
}

int item_open_action_cb(Ihandle* item_open)
{
  Ihandle *filedlg;
  Ihandle* config;
  const char* dir;

  if (!save_check(item_open))
    return IUP_DEFAULT;

  config = (Ihandle*)IupGetAttribute(item_open, "CONFIG");
  dir = IupConfigGetVariableStr(config, "MainWindow", "LastDirectory");
```

```
  filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "OPEN");
  IupSetAttribute(filedlg, "EXTFILTER", "Text␣Files|*.txt|All␣Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_open));
  IupSetStrAttribute(filedlg, "DIRECTORY", dir);

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    open_file(item_open, filename);

    dir = IupGetAttribute(filedlg, "DIRECTORY");
    IupConfigSetVariableStr(config, "MainWindow", "LastDirectory", dir);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_saveas_action_cb(Ihandle* item_saveas)
{
  Ihandle* multitext = IupGetDialogChild(item_saveas, "MULTITEXT");
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
  const char* dir = IupConfigGetVariableStr(config, "MainWindow", "LastDirectory");
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "SAVE");
  IupSetAttribute(filedlg, "EXTFILTER", "Text␣Files|*.txt|All␣Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_saveas));
  IupSetStrAttribute(filedlg, "FILE", IupGetAttribute(multitext, "FILENAME"));
  IupSetStrAttribute(filedlg, "DIRECTORY", dir);

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    saveas_file(multitext, filename);

    dir = IupGetAttribute(filedlg, "DIRECTORY");
    IupConfigSetVariableStr(config, "MainWindow", "LastDirectory", dir);
  }
```

```
  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_save_action_cb(Ihandle* item_save)
{
  Ihandle* multitext = IupGetDialogChild(item_save, "MULTITEXT");
  char* filename = IupGetAttribute(multitext, "FILENAME");
  if (!filename)
    item_saveas_action_cb(item_save);
  else
  {
    /* test again because in can be called using the hot key */
    int dirty = IupGetInt(multitext, "DIRTY");
    if (dirty)
      save_file(multitext);
  }
  return IUP_DEFAULT;
}

int item_revert_action_cb(Ihandle* item_revert)
{
  Ihandle* multitext = IupGetDialogChild(item_revert, "MULTITEXT");
  char* filename = IupGetAttribute(multitext, "FILENAME");
  open_file(item_revert, filename);
  return IUP_DEFAULT;
}

int item_exit_action_cb(Ihandle* item_exit)
{
  Ihandle* dlg = IupGetDialog(item_exit);
  Ihandle* config = (Ihandle*)IupGetAttribute(dlg, "CONFIG");

  if (!save_check(item_exit))
    return IUP_IGNORE;  /* to abort the CLOSE_CB callback */

  IupConfigDialogClosed(config, dlg, "MainWindow");
  IupConfigSave(config);
  IupDestroy(config);
  return IUP_CLOSE;
}
```

```
int goto_ok_action_cb(Ihandle* bt_ok)
{
  int line_count = IupGetInt(bt_ok, "TEXT_LINECOUNT");
  Ihandle* txt = IupGetDialogChild(bt_ok, "LINE_TEXT");
  int line = IupGetInt(txt, "VALUE");
  if (line < 1 || line >= line_count)
  {
    IupMessage("Error", "Invalid line number.");
    return IUP_DEFAULT;
  }

  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "1");
  return IUP_CLOSE;
}

int goto_cancel_action_cb(Ihandle* bt_ok)
{
  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "0");
  return IUP_CLOSE;
}

int item_goto_action_cb(Ihandle* item_goto)
{
  Ihandle* multitext = IupGetDialogChild(item_goto, "MULTITEXT");
  Ihandle *goto_dlg, *box, *bt_ok, *bt_cancel, *txt, *lbl;
  int line_count = IupGetInt(multitext, "LINECOUNT");

  lbl = IupLabel(NULL);
  IupSetfAttribute(lbl, "TITLE", "Line Number [1-%d]:", line_count);
  txt = IupText(NULL);
  IupSetAttribute(txt, "MASK", IUP_MASK_UINT);  /* unsigned integer numbers only */
  IupSetAttribute(txt, "NAME", "LINE_TEXT");
  IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
  bt_ok = IupButton("OK", NULL);
  IupSetInt(bt_ok, "TEXT_LINECOUNT", line_count);
  IupSetAttribute(bt_ok, "PADDING", "10x2");
  IupSetCallback(bt_ok, "ACTION", (Icallback)goto_ok_action_cb);
  bt_cancel = IupButton("Cancel", NULL);
  IupSetCallback(bt_cancel, "ACTION", (Icallback)goto_cancel_action_cb);
  IupSetAttribute(bt_cancel, "PADDING", "10x2");
```

```
  box = IupVbox(
    lbl,
    txt,
    IupSetAttributes(IupHbox(
      IupFill(),
      bt_ok,
      bt_cancel,
      NULL), "NORMALIZESIZE=HORIZONTAL"),
    NULL);
  IupSetAttribute(box, "MARGIN", "10x10");
  IupSetAttribute(box, "GAP", "5");

  goto_dlg = IupDialog(box);
  IupSetAttribute(goto_dlg, "TITLE", "Go To Line");
  IupSetAttribute(goto_dlg, "DIALOGFRAME", "Yes");
  IupSetAttributeHandle(goto_dlg, "DEFAULTENTER", bt_ok);
  IupSetAttributeHandle(goto_dlg, "DEFAULTESC", bt_cancel);
  IupSetAttributeHandle(goto_dlg, "PARENTDIALOG", IupGetDialog(item_goto));

  IupPopup(goto_dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(goto_dlg, "STATUS") == 1)
  {
    int line = IupGetInt(txt, "VALUE");
    int pos;
    IupTextConvertLinColToPos(multitext, line, 0, &pos);
    IupSetInt(multitext, "CARETPOS", pos);
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }

  IupDestroy(goto_dlg);

  return IUP_DEFAULT;
}

int find_next_action_cb(Ihandle* ih)
{
  /* this callback can be called from the main dialog also */
  Ihandle* find_dlg = (Ihandle*)IupGetAttribute(ih, "FIND_DIALOG");
  if (find_dlg)
  {
    char* str;
```

```
  int pos;
  Ihandle* multitext = (Ihandle*)IupGetAttribute(find_dlg, "MULTITEXT");
  int find_pos = IupGetInt(multitext, "FIND_POS");

  Ihandle* txt = IupGetDialogChild(find_dlg, "FIND_TEXT");
  char* str_to_find = IupGetAttribute(txt, "VALUE");

  Ihandle* find_case = IupGetDialogChild(find_dlg, "FIND_CASE");
  int casesensitive = IupGetInt(find_case, "VALUE");

  /* test again, because it can be called from the hot key */
  if (!str_to_find || str_to_find[0] == 0)
    return IUP_DEFAULT;

  if (find_pos == -1)
    find_pos = 0;

  str = IupGetAttribute(multitext, "VALUE");

  pos = str_find(str + find_pos, str_to_find, casesensitive);
  if (pos >= 0)
    pos += find_pos;
  else if (find_pos > 0)
    pos = str_find(str, str_to_find, casesensitive);  /* try again from the start

  if (pos >= 0)
  {
    int lin, col,
      end_pos = pos + (int)strlen(str_to_find);

    IupSetInt(multitext, "FIND_POS", end_pos);

    IupSetFocus(multitext);
    IupSetfAttribute(multitext, "SELECTIONPOS", "%d:%d", pos, end_pos);
    IupSetfAttribute(multitext, "FIND_SELECTION", "%d:%d", pos, end_pos);

    IupTextConvertPosToLinCol(multitext, pos, &lin, &col);
    IupTextConvertLinColToPos(multitext, lin, 0, &pos);  /* position at col=0, ju
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }
  else
  {
```

```
      IupSetInt(multitext, "FIND_POS", -1);
      IupMessage("Warning", "Text not found.");
    }
  }

  return IUP_DEFAULT;
}

int find_replace_action_cb(Ihandle* bt_replace)
{
  Ihandle* find_dlg = (Ihandle*)IupGetAttribute(bt_replace, "FIND_DIALOG");
  Ihandle* multitext = (Ihandle*)IupGetAttribute(find_dlg, "MULTITEXT");
  int find_pos = IupGetInt(multitext, "FIND_POS");
  char* selectionpos = IupGetAttribute(multitext, "SELECTIONPOS");
  char* find_selection = IupGetAttribute(multitext, "FIND_SELECTION");

  if (find_pos == -1 || !selectionpos || !find_selection || strcmp(selectionpos, fi
    find_next_action_cb(bt_replace);
  else
  {
    Ihandle* replace_txt = IupGetDialogChild(find_dlg, "REPLACE_TEXT");
    char* str_to_replace = IupGetAttribute(replace_txt, "VALUE");
    IupSetAttribute(multitext, "SELECTEDTEXT", str_to_replace);

    /* then find next */
    find_next_action_cb(bt_replace);
  }

  return IUP_DEFAULT;
}

int find_close_action_cb(Ihandle* bt_close)
{
  Ihandle* find_dlg = IupGetDialog(bt_close);
  Ihandle* multitext = (Ihandle*)IupGetAttribute(find_dlg, "MULTITEXT");
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
  IupConfigDialogClosed(config, find_dlg, "FindDialog");
  IupHide(find_dlg);  /* do not destroy, just hide */
  return IUP_DEFAULT;
}

Ihandle* create_find_dialog(Ihandle *multitext)
```

```
{
  Ihandle *box, *bt_next, *bt_close, *txt, *find_case, *find_dlg;
  Ihandle* txt_replace, *bt_replace;

  txt = IupText(NULL);
  IupSetAttribute(txt, "NAME", "FIND_TEXT");
  IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
  txt_replace = IupText(NULL);
  IupSetAttribute(txt_replace, "NAME", "REPLACE_TEXT");
  IupSetAttribute(txt_replace, "VISIBLECOLUMNS", "20");
  find_case = IupToggle("Case Sensitive", NULL);
  IupSetAttribute(find_case, "NAME", "FIND_CASE");
  bt_next = IupButton("Find Next", NULL);
  IupSetAttribute(bt_next, "PADDING", "10x2");
  IupSetCallback(bt_next, "ACTION", (Icallback)find_next_action_cb);
  bt_replace = IupButton("Replace", NULL);
  IupSetAttribute(bt_replace, "PADDING", "10x2");
  IupSetCallback(bt_replace, "ACTION", (Icallback)find_replace_action_cb);
  IupSetAttribute(bt_replace, "NAME", "REPLACE_BUTTON");
  bt_close = IupButton("Close", NULL);
  IupSetCallback(bt_close, "ACTION", (Icallback)find_close_action_cb);
  IupSetAttribute(bt_close, "PADDING", "10x2");

  box = IupVbox(
    IupLabel("Find What:"),
    txt,
    IupSetAttributes(IupLabel("Replace with:"), "NAME=REPLACE_LABEL"),
    txt_replace,
    find_case,
    IupSetAttributes(IupHbox(
      IupFill(),
      bt_next,
      bt_replace,
      bt_close,
      NULL), "NORMALIZESIZE=HORIZONTAL"),
    NULL);
  IupSetAttribute(box, "MARGIN", "10x10");
  IupSetAttribute(box, "GAP", "5");

  find_dlg = IupDialog(box);
  IupSetAttribute(find_dlg, "TITLE", "Find");
  IupSetAttribute(find_dlg, "DIALOGFRAME", "Yes");
```

```
  IupSetAttributeHandle(find_dlg, "DEFAULTENTER", bt_next);
  IupSetAttributeHandle(find_dlg, "DEFAULTESC", bt_close);
  IupSetAttributeHandle(find_dlg, "PARENTDIALOG", IupGetDialog(multitext));
  IupSetCallback(find_dlg, "CLOSE_CB", (Icallback)find_close_action_cb);

  /* Save the multiline to access it from the callbacks */
  IupSetAttribute(find_dlg, "MULTITEXT", (char*)multitext);

  /* Save the dialog to reuse it */
  IupSetAttribute(find_dlg, "FIND_DIALOG", (char*)find_dlg);  /* from itself */
  IupSetAttribute(IupGetDialog(multitext), "FIND_DIALOG", (char*)find_dlg); /* from

  return find_dlg;
}

int item_find_action_cb(Ihandle* item_find)
{
  Ihandle* find_dlg = (Ihandle*)IupGetAttribute(item_find, "FIND_DIALOG");
  Ihandle* multitext = IupGetDialogChild(item_find, "MULTITEXT");
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
  char* str;

  if (!find_dlg)
    find_dlg = create_find_dialog(multitext);

  set_find_replace_visibility(find_dlg, 0);

  IupConfigDialogShow(config, find_dlg, "FindDialog");

  str = IupGetAttribute(multitext, "SELECTEDTEXT");
  if (str && str[0] != 0)
  {
    Ihandle* txt = IupGetDialogChild(find_dlg, "FIND_TEXT");
    IupSetStrAttribute(txt, "VALUE", str);
  }

  return IUP_DEFAULT;
}

int item_replace_action_cb(Ihandle* item_replace)
{
  Ihandle* find_dlg = (Ihandle*)IupGetAttribute(item_replace, "FIND_DIALOG");
```

```
  Ihandle* multitext = IupGetDialogChild(item_replace, "MULTITEXT");
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
  char* str;

  if (!find_dlg)
    find_dlg = create_find_dialog(multitext);

  set_find_replace_visibility(find_dlg, 1);

  IupConfigDialogShow(config, find_dlg, "FindDialog");

  str = IupGetAttribute(multitext, "SELECTEDTEXT");
  if (str && str[0] != 0)
  {
    Ihandle* txt = IupGetDialogChild(find_dlg, "FIND_TEXT");
    IupSetStrAttribute(txt, "VALUE", str);
  }

  return IUP_IGNORE;  /* replace system processing for the hot key */
}

int selection_find_next_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  char* str = IupGetAttribute(multitext, "SELECTEDTEXT");
  if (str && str[0] != 0)
  {
    Ihandle* txt;
    Ihandle* find_dlg = (Ihandle*)IupGetAttribute(ih, "FIND_DIALOG");

    if (!find_dlg)
      find_dlg = create_find_dialog(multitext);

    txt = IupGetDialogChild(find_dlg, "FIND_TEXT");
    IupSetStrAttribute(txt, "VALUE", str);

    find_next_action_cb(ih);
  }

  return IUP_DEFAULT;
}
```

```
int item_copy_action_cb(Ihandle* item_copy)
{
  Ihandle* multitext = IupGetDialogChild(item_copy, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(clipboard, "TEXT", IupGetAttribute(multitext, "SELECTEDTEXT"));
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_paste_action_cb(Ihandle* item_paste)
{
  Ihandle* multitext = IupGetDialogChild(item_paste, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(multitext, "INSERT", IupGetAttribute(clipboard, "TEXT"));
  IupDestroy(clipboard);
  return IUP_IGNORE;  /* replace system processing for the hot key, to correctly pa
}

int item_cut_action_cb(Ihandle* item_cut)
{
  Ihandle* multitext = IupGetDialogChild(item_cut, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(clipboard, "TEXT", IupGetAttribute(multitext, "SELECTEDTEXT"));
  IupSetAttribute(multitext, "SELECTEDTEXT", "");
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_delete_action_cb(Ihandle* item_delete)
{
  Ihandle* multitext = IupGetDialogChild(item_delete, "MULTITEXT");
  IupSetAttribute(multitext, "SELECTEDTEXT", "");
  return IUP_DEFAULT;
}

int item_select_all_action_cb(Ihandle* item_select_all)
{
  Ihandle* multitext = IupGetDialogChild(item_select_all, "MULTITEXT");
  IupSetFocus(multitext);
  IupSetAttribute(multitext, "SELECTION", "ALL");
  return IUP_DEFAULT;
```

```
}

int item_font_action_cb(Ihandle* item_font)
{
  Ihandle* multitext = IupGetDialogChild(item_font, "MULTITEXT");
  Ihandle* fontdlg = IupFontDlg();
  char* font = IupGetAttribute(multitext, "FONT");
  IupSetStrAttribute(fontdlg, "VALUE", font);
  IupSetAttributeHandle(fontdlg, "PARENTDIALOG", IupGetDialog(item_font));

  IupPopup(fontdlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(fontdlg, "STATUS") == 1)
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
    font = IupGetAttribute(fontdlg, "VALUE");
    IupSetStrAttribute(multitext, "FONT", font);

    IupConfigSetVariableStr(config, "MainWindow", "Font", font);
  }

  IupDestroy(fontdlg);
  return IUP_DEFAULT;
}

int item_toolbar_action_cb(Ihandle* item_toolbar)
{
  Ihandle* multitext = IupGetDialogChild(item_toolbar, "MULTITEXT");
  Ihandle* toolbar = IupGetChild(IupGetParent(multitext), 0);
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

  toggle_bar_visibility(item_toolbar, toolbar);

  IupConfigSetVariableStr(config, "MainWindow", "Toolbar", IupGetAttribute(item_too
  return IUP_DEFAULT;
}

int item_statusbar_action_cb(Ihandle* item_statusbar)
{
  Ihandle* multitext = IupGetDialogChild(item_statusbar, "MULTITEXT");
  Ihandle* statusbar = IupGetBrother(multitext);
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
```

```
  toggle_bar_visibility(item_statusbar, statusbar);

  IupConfigSetVariableStr(config, "MainWindow", "Statusbar", IupGetAttribute(item_s
  return IUP_DEFAULT;
}

int item_help_action_cb(void)
{
  IupHelp("http://www.tecgraf.puc-rio.br/iup");
  return IUP_DEFAULT;
}

int item_about_action_cb(void)
{
  IupMessage("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣S
  return IUP_DEFAULT;
}


/********************************** Main ****************************************/


Ihandle* create_main_dialog(Ihandle *config)
{
  Ihandle *dlg, *vbox, *multitext, *menu;
  Ihandle *sub_menu_file, *file_menu, *item_exit, *item_new, *item_open, *item_save
  Ihandle *sub_menu_edit, *edit_menu, *item_find, *item_find_next, *item_goto, *iten
  Ihandle *btn_cut, *btn_copy, *btn_paste, *btn_find, *btn_new, *btn_open, *btn_sav
  Ihandle *sub_menu_format, *format_menu, *item_font, *item_replace;
  Ihandle *sub_menu_help, *help_menu, *item_help, *item_about;
  Ihandle *sub_menu_view, *view_menu, *item_toolbar, *item_statusbar;
  Ihandle *lbl_statusbar, *toolbar_hb, *recent_menu;
  const char* font;

  multitext = IupText(NULL);
  IupSetAttribute(multitext, "MULTILINE", "YES");
  IupSetAttribute(multitext, "EXPAND", "YES");
  IupSetAttribute(multitext, "NAME", "MULTITEXT");
  IupSetAttribute(multitext, "DIRTY", "NO");
  IupSetCallback(multitext, "CARET_CB", (Icallback)multitext_caret_cb);
  IupSetCallback(multitext, "VALUECHANGED_CB", (Icallback)multitext_valuechanged_cb
```

```
IupSetCallback(multitext, "DROPFILES_CB", (Icallback)dropfiles_cb);

lbl_statusbar = IupLabel("Lin 1, Col 1");
IupSetAttribute(lbl_statusbar, "NAME", "STATUSBAR");
IupSetAttribute(lbl_statusbar, "EXPAND", "HORIZONTAL");
IupSetAttribute(lbl_statusbar, "PADDING", "10x5");

item_new = IupItem("&New\tCtrl+N", NULL);
IupSetAttribute(item_new, "IMAGE", "IUP_FileNew");
IupSetCallback(item_new, "ACTION", (Icallback)item_new_action_cb);
btn_new = IupButton(NULL, NULL);
IupSetAttribute(btn_new, "IMAGE", "IUP_FileNew");
IupSetAttribute(btn_new, "FLAT", "Yes");
IupSetCallback(btn_new, "ACTION", (Icallback)item_new_action_cb);
IupSetAttribute(btn_new, "TIP", "New (Ctrl+N)");
IupSetAttribute(btn_new, "CANFOCUS", "No");

item_open = IupItem("&Open...\tCtrl+O", NULL);
IupSetAttribute(item_open, "IMAGE", "IUP_FileOpen");
IupSetCallback(item_open, "ACTION", (Icallback)item_open_action_cb);
btn_open = IupButton(NULL, NULL);
IupSetAttribute(btn_open, "IMAGE", "IUP_FileOpen");
IupSetAttribute(btn_open, "FLAT", "Yes");
IupSetCallback(btn_open, "ACTION", (Icallback)item_open_action_cb);
IupSetAttribute(btn_open, "TIP", "Open (Ctrl+O)");
IupSetAttribute(btn_open, "CANFOCUS", "No");

item_save = IupItem("&Save\tCtrl+S", NULL);
IupSetAttribute(item_save, "NAME", "ITEM_SAVE");
IupSetAttribute(item_save, "IMAGE", "IUP_FileSave");
IupSetCallback(item_save, "ACTION", (Icallback)item_save_action_cb);
btn_save = IupButton(NULL, NULL);
IupSetAttribute(btn_save, "IMAGE", "IUP_FileSave");
IupSetAttribute(btn_save, "FLAT", "Yes");
IupSetCallback(btn_save, "ACTION", (Icallback)item_save_action_cb);
IupSetAttribute(btn_save, "TIP", "Save (Ctrl+S)");
IupSetAttribute(btn_save, "CANFOCUS", "No");

item_saveas = IupItem("Save &As...", NULL);
IupSetAttribute(item_saveas, "NAME", "ITEM_SAVEAS");
IupSetCallback(item_saveas, "ACTION", (Icallback)item_saveas_action_cb);
```

```
item_revert = IupItem("&Revert", NULL);
IupSetAttribute(item_revert, "NAME", "ITEM_REVERT");
IupSetCallback(item_revert, "ACTION", (Icallback)item_revert_action_cb);

item_exit = IupItem("E&xit", NULL);
IupSetCallback(item_exit, "ACTION", (Icallback)item_exit_action_cb);

item_find = IupItem("&Find...\tCtrl+F", NULL);
IupSetAttribute(item_find, "IMAGE", "IUP_EditFind");
IupSetCallback(item_find, "ACTION", (Icallback)item_find_action_cb);
btn_find = IupButton(NULL, NULL);
IupSetAttribute(btn_find, "IMAGE", "IUP_EditFind");
IupSetAttribute(btn_find, "FLAT", "Yes");
IupSetCallback(btn_find, "ACTION", (Icallback)item_find_action_cb);
IupSetAttribute(btn_find, "TIP", "Find (Ctrl+F)");
IupSetAttribute(btn_find, "CANFOCUS", "No");

item_find_next = IupItem("Find &Next\tF3", NULL);
IupSetAttribute(item_find_next, "NAME", "ITEM_FINDNEXT");
IupSetCallback(item_find_next, "ACTION", (Icallback)find_next_action_cb);

item_replace = IupItem("&Replace...\tCtrl+H", NULL);
IupSetCallback(item_replace, "ACTION", (Icallback)item_replace_action_cb);

item_cut = IupItem("Cu&t\tCtrl+X", NULL);
IupSetAttribute(item_cut, "NAME", "ITEM_CUT");
IupSetAttribute(item_cut, "IMAGE", "IUP_EditCut");
IupSetCallback(item_cut, "ACTION", (Icallback)item_cut_action_cb);
btn_cut = IupButton(NULL, NULL);
IupSetAttribute(btn_cut, "IMAGE", "IUP_EditCut");
IupSetAttribute(btn_cut, "FLAT", "Yes");
IupSetCallback(btn_cut, "ACTION", (Icallback)item_cut_action_cb);
IupSetAttribute(btn_cut, "TIP", "Cut (Ctrl+X)");
IupSetAttribute(btn_cut, "CANFOCUS", "No");

item_copy = IupItem("&Copy\tCtrl+C", NULL);
IupSetAttribute(item_copy, "NAME", "ITEM_COPY");
IupSetAttribute(item_copy, "IMAGE", "IUP_EditCopy");
IupSetCallback(item_copy, "ACTION", (Icallback)item_copy_action_cb);
btn_copy = IupButton(NULL, NULL);
IupSetAttribute(btn_copy, "IMAGE", "IUP_EditCopy");
IupSetAttribute(btn_copy, "FLAT", "Yes");
```

```
IupSetCallback(btn_copy, "ACTION", (Icallback)item_copy_action_cb);
IupSetAttribute(btn_copy, "TIP", "Copy (Ctrl+C)");
IupSetAttribute(btn_copy, "CANFOCUS", "No");

item_paste = IupItem("&Paste\tCtrl+V", NULL);
IupSetAttribute(item_paste, "NAME", "ITEM_PASTE");
IupSetAttribute(item_paste, "IMAGE", "IUP_EditPaste");
IupSetCallback(item_paste, "ACTION", (Icallback)item_paste_action_cb);
btn_paste = IupButton(NULL, NULL);
IupSetAttribute(btn_paste, "IMAGE", "IUP_EditPaste");
IupSetAttribute(btn_paste, "FLAT", "Yes");
IupSetCallback(btn_paste, "ACTION", (Icallback)item_paste_action_cb);
IupSetAttribute(btn_paste, "TIP", "Paste (Ctrl+V)");
IupSetAttribute(btn_paste, "CANFOCUS", "No");

item_delete = IupItem("&Delete\tDel", NULL);
IupSetAttribute(item_delete, "IMAGE", "IUP_EditErase");
IupSetAttribute(item_delete, "NAME", "ITEM_DELETE");
IupSetCallback(item_delete, "ACTION", (Icallback)item_delete_action_cb);

item_select_all = IupItem("Select &All\tCtrl+A", NULL);
IupSetCallback(item_select_all, "ACTION", (Icallback)item_select_all_action_cb);

item_goto = IupItem("&Go To...\tCtrl+G", NULL);
IupSetCallback(item_goto, "ACTION", (Icallback)item_goto_action_cb);

item_toolbar = IupItem("&Toobar", NULL);
IupSetCallback(item_toolbar, "ACTION", (Icallback)item_toolbar_action_cb);
IupSetAttribute(item_toolbar, "VALUE", "ON");

item_statusbar = IupItem("&Statusbar", NULL);
IupSetCallback(item_statusbar, "ACTION", (Icallback)item_statusbar_action_cb);
IupSetAttribute(item_statusbar, "VALUE", "ON");

item_font = IupItem("&Font...", NULL);
IupSetCallback(item_font, "ACTION", (Icallback)item_font_action_cb);

item_help = IupItem("&Help...", NULL);
IupSetCallback(item_help, "ACTION", (Icallback)item_help_action_cb);

item_about = IupItem("&About...", NULL);
IupSetCallback(item_about, "ACTION", (Icallback)item_about_action_cb);
```

```
  recent_menu = IupMenu(NULL);

  file_menu = IupMenu(
    item_new,
    item_open,
    item_save,
    item_saveas,
    item_revert,
    IupSeparator(),
    IupSubmenu("Recent␣&Files", recent_menu),
    item_exit,
    NULL);
  edit_menu = IupMenu(
    item_cut,
    item_copy,
    item_paste,
    item_delete,
    IupSeparator(),
    item_find,
    item_find_next,
    item_replace,
    item_goto,
    IupSeparator(),
    item_select_all,
    NULL);
  format_menu = IupMenu(
    item_font,
    NULL);
  view_menu = IupMenu(
    item_toolbar,
    item_statusbar,
    NULL);
  help_menu = IupMenu(
    item_help,
    item_about,
    NULL);

  IupSetCallback(file_menu, "OPEN_CB", (Icallback)file_menu_open_cb);
  IupSetCallback(edit_menu, "OPEN_CB", (Icallback)edit_menu_open_cb);

  sub_menu_file = IupSubmenu("&File", file_menu);
```

```
sub_menu_edit = IupSubmenu("&Edit", edit_menu);
sub_menu_format = IupSubmenu("F&ormat", format_menu);
sub_menu_view = IupSubmenu("&View", view_menu);
sub_menu_help = IupSubmenu("&Help", help_menu);

menu = IupMenu(
  sub_menu_file,
  sub_menu_edit,
  sub_menu_format,
  sub_menu_view,
  sub_menu_help,
  NULL);

toolbar_hb = IupHbox(
  btn_new,
  btn_open,
  btn_save,
  IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
  btn_cut,
  btn_copy,
  btn_paste,
  IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
  btn_find,
  NULL);
IupSetAttribute(toolbar_hb, "MARGIN", "5x5");
IupSetAttribute(toolbar_hb, "GAP", "2");

vbox = IupVbox(
  toolbar_hb,
  multitext,
  lbl_statusbar,
  NULL);

dlg = IupDialog(vbox);
IupSetAttributeHandle(dlg, "MENU", menu);
IupSetAttribute(dlg, "SIZE", "HALFxHALF");
IupSetCallback(dlg, "CLOSE_CB", (Icallback)item_exit_action_cb);
IupSetCallback(dlg, "DROPFILES_CB", (Icallback)dropfiles_cb);

IupSetCallback(dlg, "K_cN", (Icallback)item_new_action_cb);
IupSetCallback(dlg, "K_cO", (Icallback)item_open_action_cb);
IupSetCallback(dlg, "K_cS", (Icallback)item_save_action_cb);
```

```
  IupSetCallback(dlg, "K_cF", (Icallback)item_find_action_cb);
  IupSetCallback(dlg, "K_cH", (Icallback)item_replace_action_cb);  /* replace syster
  IupSetCallback(dlg, "K_cG", (Icallback)item_goto_action_cb);
  IupSetCallback(dlg, "K_F3", (Icallback)find_next_action_cb);
  IupSetCallback(dlg, "K_cF3", (Icallback)selection_find_next_action_cb);
  IupSetCallback(dlg, "K_cV", (Icallback)item_paste_action_cb);  /* replace system
  /* Ctrl+C, Ctrl+X, Ctrl+A, Del, already implemented inside IupText */

  /* parent for pre-defined dialogs in closed functions (IupMessage and IupAlarm) *
  IupSetAttributeHandle(NULL, "PARENTDIALOG", dlg);

  /* Initialize variables from the configuration file */

  IupConfigRecentInit(config, recent_menu, config_recent_cb, 10);

  font = IupConfigGetVariableStr(config, "MainWindow", "Font");
  if (font)
    IupSetStrAttribute(multitext, "FONT", font);

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Toolbar", 1))
  {
    IupSetAttribute(item_toolbar, "VALUE", "OFF");

    IupSetAttribute(toolbar_hb, "FLOATING", "YES");
    IupSetAttribute(toolbar_hb, "VISIBLE", "NO");
  }

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Statusbar", 1))
  {
    IupSetAttribute(item_statusbar, "VALUE", "OFF");

    IupSetAttribute(lbl_statusbar, "FLOATING", "YES");
    IupSetAttribute(lbl_statusbar, "VISIBLE", "NO");
  }

  IupSetAttribute(dlg, "CONFIG", (char*)config);

  return dlg;
}

int main(int argc, char **argv)
{
```

```
  Ihandle *dlg;
  Ihandle *config;

  IupOpen(&argc, &argv);
  IupImageLibOpen();

  config = IupConfig();
  IupSetAttribute(config, "APP_NAME", "simple_notepad");
  IupConfigLoad(config);

  dlg = create_main_dialog(config);

  /* show the dialog at the last position, with the last size */
  IupConfigDialogShow(config, dlg, "MainWindow");

  /* initialize the current file */
  new_file(dlg);

  /* open a file from the command line (allow file association in Windows) */
  if (argc > 1 && argv[1])
  {
    const char* filename = argv[1];
    open_file(dlg, filename);
  }

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}

/* If instead of using IupText we use IupScintilla, then we can add:
   - more find/replace options
   - zoom
   - show white spaces
   - margins
   - word wrap
   - tab size
   - auto replace tabs by spaces
   - undo & redo
   - markers
   - line numbers
```

```
   and much more.
   Hot keys for:
   - match braces
   - to lower case
   - to upper case
 */

Lua语言
require("iuplua")
require("iupluaimglib")


--******************************** Utilities *********************************

function str_nocase(s)
  s = string.gsub(s, "%a", function (c)
      return string.format("[%s%s]", string.lower(c),
                                     string.upper(c))
    end)
  return s
end

function str_find(str, str_to_find, casesensitive, start)
  if (not casesensitive) then
    str_to_find = str_nocase(str_to_find)
    return string.find(str, str_to_find, start)
  end

  return string.find(str, str_to_find, start, true)
end

function str_splitfilename(filename)
  return string.match(filename, "(.-)([^\\/]-%.?([^%.\\/]*))$")
end

function str_filetitle(filename)
  local path, title, ext = str_splitfilename(filename)
  return title
end

function read_file(filename)
  local ifile = io.open(filename, "r")
```

```
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return nil
  end

  local str = ifile:read("*a")
  if (not str) then
    iup.Message("Error", "Fail when reading from file: " .. filename)
    return nil
  end

  ifile:close()
  return str
end

function write_file(filename, str)
  local ifile = io.open(filename, "w")
  if (not ifile) then
    iup.Message("Error", "Can't open file: " .. filename)
    return false
  end

  if (not ifile:write(str)) then
    iup.Message("Error", "Fail when writing to file: " .. filename)
  end

  ifile:close()
  return true
end

function new_file(ih)
  local dlg = iup.GetDialog(ih)
  local multitext = dlg.multitext

  dlg.title = "Untitled - Simple Notepad"
  multitext.filename = nil
  multitext.dirty = nil
  multitext.value = ""
end

function open_file(ih, filename)
  local str = read_file(filename)
```

```
  if (str) then
    local dlg = iup.GetDialog(ih)
    local multitext = dlg.multitext
    local config = multitext.config

    dlg.title = str_filetitle(filename).." - Simple Notepad"
    multitext.filename = filename
    multitext.dirty = nil
    multitext.value = str

    config:RecentUpdate(filename)
  end
end

function save_file(multitext)
  if (write_file(multitext.filename, multitext.value)) then
    multitext.dirty = nil
  end
end

function saveas_file(multitext, filename)
  if (write_file(filename, multitext.value)) then
    local dlg = iup.GetDialog(multitext)
    local config = multitext.config

    dlg.title = str_filetitle(filename).." - Simple Notepad"
    multitext.filename = filename
    multitext.dirty = nil

    config:RecentUpdate(filename)
  end
end

function save_check(ih)
  local dlg = iup.GetDialog(ih)
  local multitext = dlg.multitext

  if (multitext.dirty) then
    local resp = iup.Alarm("Warning", "File not saved! Save it now?", "Yes", "No",
    if resp == 1 then -- save the changes and continue
      save_file(multitext)
    elseif resp == 3 then  -- cancel
```

```
      return false
    else  -- ignore the changes and continue
    end
  end
  return true
end

function toggle_bar_visibility(item, bar)
  if (item.value == "ON") then
    bar.floating = "YES"
    bar.visible = "NO"
    item.value = "OFF"
  else
    bar.floating = "NO"
    bar.visible = "YES"
    item.value = "ON"
  end
  iup.Refresh(bar)  -- refresh the dialog layout
end

function set_find_replace_visibility(find_dlg, show_replace)
  local replace_txt = find_dlg.replace_txt
  local replace_lbl = find_dlg.replace_lbl
  local replace_bt = find_dlg.replace_bt

  if (show_replace) then
    replace_txt.visible = "Yes"
    replace_lbl.visible = "Yes"
    replace_bt.visible = "Yes"
    replace_txt.floating = "No"
    replace_lbl.floating = "No"
    replace_bt.floating  = "No"

    find_dlg.title = "Replace"
  else
    replace_txt.floating = "Yes"
    replace_lbl.floating = "Yes"
    replace_bt.floating = "Yes"
    replace_txt.visible = "No"
    replace_lbl.visible = "No"
    replace_bt.visible = "No"
```

```
    find_dlg.title = "Find"
  end

  find_dlg.size = nil  -- force a dialog resize on the iup.Refresh
  iup.Refresh(find_dlg)
end


--********************************* Main (Part 1/2) ****************************

-- create all the elements that will have callbacks in Lua prior to callbacks defin

config = iup.config{}
config.app_name = "simple_notepad"
config:Load()

lbl_statusbar = iup.label{title = "Lin 1, Col 1", expand = "HORIZONTAL", padding =

multitext = iup.text{
  multiline = "YES",
  expand = "YES",
  config = config,
  dirty = nil,
}

item_new = iup.item{title = "&New...\tCtrl+N", image = "IUP_FileNew"}
item_open = iup.item{title = "&Open...\tCtrl+O", image = "IUP_FileOpen"}
item_save = iup.item{title="&Save\tCtrl+S"}
item_saveas = iup.item{title="Save &As...", image = "IUP_FileSave"}
item_revert = iup.item{title="&Revert"}
item_exit = iup.item{title="E&xit"}
item_find = iup.item{title="&Find...\tCtrl+F", image = "IUP_EditFind"}
item_find_next = iup.item{title = "Find &Next\tF3"}
item_replace = iup.item{title = "&Replace...\tCtrl+H"}
item_copy = iup.item{title="&Copy\tCtrl+C", image = "IUP_EditCopy"}
item_paste = iup.item{title="&Paste\tCtrl+V", image = "IUP_EditPaste"}
item_cut = iup.item{title="Cu&t\tCtrl+X", image = "IUP_EditCut"}
item_delete = iup.item{title="&Delete\tDel", image = "IUP_EditErase"}
item_select_all = iup.item{title="Select &All\tCtrl+A"}
item_goto = iup.item{title="&Go To...\tCtrl+G"}
item_font = iup.item{title="&Font..."}
item_toolbar = iup.item{title="&Toobar", value="ON"}
```

```
item_statusbar = iup.item{title="&Statusbar", value="ON"}
item_help = iup.item{title="&Help..."}
item_about = iup.item{title="&About..."}

recent_menu = iup.menu{}

file_menu = iup.menu{
  item_new,
  item_open,
  item_save,
  item_saveas,
  item_revert,
  iup.separator{},
  iup.submenu{title="Recent &Files", recent_menu},
  item_exit
  }

edit_menu = iup.menu{
  item_cut,
  item_copy,
  item_paste,
  item_delete,
  iup.separator{},
  item_find,
  item_find_next,
  item_replace,
  item_goto,
  iup.separator{},
  item_select_all
  }

format_menu = iup.menu{item_font}
view_menu = iup.menu{item_toolbar, item_statusbar}
help_menu = iup.menu{item_help, item_about}

sub_menu_file = iup.submenu{file_menu, title = "&File"}
sub_menu_edit = iup.submenu{edit_menu, title = "&Edit"}
sub_menu_format = iup.submenu{format_menu, title = "F&ormat"}
sub_menu_view = iup.submenu{title = "&View", view_menu}
sub_menu_help = iup.submenu{help_menu, title = "&Help"}

menu = iup.menu{
```

```
  sub_menu_file,
  sub_menu_edit,
  sub_menu_format,
  sub_menu_view,
  sub_menu_help,
  }


--********************************* Callbacks *********************************


function multitext:dropfiles_cb(filename)
  if (save_check(self)) then
    open_file(self, filename)
  end
end

function multitext:valuechanged_cb()
  self.dirty = "YES"
end

function file_menu:open_cb()
  if (multitext.dirty) then
    item_save.active = "YES"
  else
    item_save.active = "NO"
  end
  if (multitext.dirty and multitext.filename) then
    item_revert.active = "YES"
  else
    item_revert.active = "NO"
  end
end

function edit_menu:open_cb()
  local clipboard = iup.clipboard{}
  if (not clipboard.textavailable) then
    item_paste.active = "NO"
  else
    item_paste.active = "YES"
  end
  clipboard:destroy()
```

```
  if (not multitext.selectedtext) then
    item_cut.active = "NO"
    item_delete.active = "NO"
    item_copy.active = "NO"
  else
    item_cut.active = "YES"
    item_delete.active = "YES"
    item_copy.active = "YES"
  end

  if (find_dlg) then
    local find_txt = find_dlg.find_txt
    if (find_txt) and (find_txt.value ~= "") then
      item_find_next.active = "YES"
    else
      item_find_next.active = "NO"
    end
  else
    item_find_next.active = "NO"
  end
end

function config:recent_cb()
  if (save_check(self)) then
    local filename = self.title
    open_file(self, filename)
  end
end

function multitext:caret_cb(lin, col)
  lbl_statusbar.title = "Lin "..lin.." , Col "..col
end

function item_new:action()
  if save_check(self) then
    new_file(self)
  end
end

function item_open:action()
  if not save_check(self) then
```

```
    return
  end

  local filedlg = iup.filedlg{
    dialogtype = "OPEN",
    extfilter="Text␣Files|*.txt|All␣Files|*.*|",
    parentdialog=iup.GetDialog(self),
    directory = config:GetVariable("MainWindow", "LastDirectory"),
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    open_file(self, filename)

    config:SetVariable("MainWindow", "LastDirectory", filedlg.directory)
  end

  filedlg:destroy()
end

function item_saveas:action()
  local filedlg = iup.filedlg{
    dialogtype = "SAVE",
    extfilter="Text␣Files|*.txt|All␣Files|*.*|",
    parentdialog = iup.GetDialog(self),
    file = multitext.filename,
    directory = config:GetVariable("MainWindow", "LastDirectory"),
    }

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    saveas_file(multitext, filename)

    config:SetVariable("MainWindow", "LastDirectory", filedlg.directory)
  end

  filedlg:destroy()
end
```

```
function item_save:action()
  if (not multitext.filename) then
    item_saveas:action()
  else
    -- test again because in can be called using the hot key
    if (multitext.dirty) then
      save_file(multitext)
    end
  end
end

function item_revert:action()
  open_file(self, multitext.filename)
end

function item_exit:action()
  if not save_check(self) then
    return iup.IGNORE  -- to abort the CLOSE_CB callback
  end

  config:DialogClosed(iup.GetDialog(self), "MainWindow")
  config:Save()
  config:destroy()
  return iup.CLOSE
end

function item_goto:action()
  local line_count = multitext.linecount
  local lbl_goto = iup.label{title = "Line Number [1-"..line_count.."]:"}
  local txt_goto = iup.text{mask = iup.MASK_UINT, visiblecolumns = 20} --unsigned i

  local bt_goto_ok = iup.button{title = "OK", text_linecount = 0, padding = "10x2"}
  bt_goto_ok.text_linecount = line_count
  function bt_goto_ok:action()
    local line_count = tonumber(self.text_linecount)
    local line = tonumber(txt_goto.value)
    if (line < 1 or line >= line_count) then
      iup.Message("Error", "Invalid line number.")
      return
    end
    goto_dlg.status = 1
```

```
    return iup.CLOSE
  end

  local bt_goto_cancel = iup.button{title = "Cancel", padding = "10x2"}
  function bt_goto_cancel:action()
    goto_dlg.status = 0
    return iup.CLOSE
  end

  local box = iup.vbox{
    lbl_goto,
    txt_goto,
    iup.hbox{
      iup.fill{},
      bt_goto_ok,
      bt_goto_cancel,
      normalizesize="HORIZONTAL",
    },
    margin = "10x10",
    gap = "5",
  }
  local goto_dlg = iup.dialog{
    box,
    title = "Go To Line",
    dialogframe = "Yes",
    defaultenter = bt_goto_ok,
    defaultesc = bt_goto_cancel,
    parentdialog = iup.GetDialog(self)
  }

  goto_dlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(goto_dlg.status) == 1) then
    local line = txt_goto.value
    local pos = iup.TextConvertLinColToPos(multitext, line, 0)
    multitext.caretpos = pos
    multitext.scrolltopos = pos
  end

  goto_dlg:destroy()
end
```

```
function item_find_next:action()
  -- test, because it can be called from the hot key
  if (find_dlg) then
    local find_pos = tonumber(find_dlg.find_pos)

    local find_txt = find_dlg.find_txt
    local str_to_find = find_txt.value

    local find_case = find_dlg.find_case
    local casesensitive = (find_case.value == "ON")

    -- test again, because it can be called from the hot key
    if (not str_to_find or str_to_find:len()==0) then
      return
    end

    if (not find_pos) then
      find_pos = 1
    end

    local str = multitext.value

    local pos, end_pos = str_find(str, str_to_find, casesensitive, find_pos)
    if (not pos) then
      pos, end_pos = str_find(str, str_to_find, casesensitive, 1)  -- try again fro
    end

    if (pos) and (pos > 0) then
      pos = pos - 1
      find_dlg.find_pos = end_pos

      iup.SetFocus(multitext)
      multitext.selectionpos = pos..":"..end_pos
      multitext.find_selection = pos..":"..end_pos

      local lin, col = iup.TextConvertPosToLinCol(multitext, pos)
      local pos = iup.TextConvertLinColToPos(multitext, lin, 0)  -- position at col
      multitext.scrolltopos = pos
    else
      find_dlg.find_pos = nil
      iup.Message("Warning", "Text not found: "..str_to_find)
    end
```

```
  end
end

function create_find_dialog()
  local find_txt = iup.text{visiblecolumns = "20"}
  local find_case = iup.toggle{title = "Case Sensitive"}
  local bt_find_next = iup.button{title = "Find Next", padding = "10x2"}
  local bt_find_close = iup.button{title = "Close", padding = "10x2"}
  local replace_txt = iup.text{visiblecolumns = "20"}
  local replace_bt = iup.button{title = "Replace", padding = "10x2", action = find_
  local replace_lbl = iup.label{title = "Replace with:"}

  function replace_bt:action()
    local find_pos = tonumber(find_dlg.find_pos)
    local selectionpos = multitext.selectionpos
    local find_selection = multitext.find_selection

    if (not find_pos or not selectionpos or not find_selection or (selectionpos ~=
      item_find_next:action()
    else
      local str_to_replace = replace_txt.value
      multitext.selectedtext = str_to_replace

      -- then find next
      item_find_next:action()
    end
  end

  function bt_find_next:action()
      item_find_next:action()
  end

  function bt_find_close:action()
    config:DialogClosed(find_dlg, "FindDialog")
    find_dlg:hide()  -- do not destroy, just hide
  end

  local box = iup.vbox{
    iup.label{title = "Find What:"},
    find_txt,
    replace_lbl,
    replace_txt,
```

```
    find_case,
    iup.hbox{
      iup.fill{},
      bt_find_next,
      replace_bt,
      bt_find_close,
      normalizesize="HORIZONTAL"
    },
    margin = "10x10",
    gap = "5"
  }

  find_dlg = iup.dialog{  -- create as global, not local
    box,
    title = "Find",
    dialogframe = "Yes",
    defaultenter = bt_next,
    defaultesc = bt_close,
    parentdialog = iup.GetDialog(multitext),
    close_cb = bt_find_close_action,

    find_txt = find_txt,
    find_case = find_case,
    replace_txt = replace_txt,
    replace_bt = replace_bt,
    replace_lbl = replace_lbl,
    }

  return find_dlg
end

function item_find:action()
  if (not find_dlg) then
    find_dlg = create_find_dialog()  -- create as global, not local
  end
  set_find_replace_visibility(find_dlg, false)

  config:DialogShow(find_dlg, "FindDialog")

  local str = multitext.selectedtext
  if (str and str:len()~=0) then
    local find_txt = find_dlg.find_txt
```

```
    find_txt.value = str
  end
end

function item_replace:action()
  if (not find_dlg) then
    find_dlg = create_find_dialog()  -- create as global, not local
  end
  set_find_replace_visibility(find_dlg, true)

  config:DialogShow(find_dlg, "FindDialog")

  local str = multitext.selectedtext
  if (str and str:len()~=0) then
    local find_txt = find_dlg.find_txt
    find_txt.value = str
  end
end

function selection_find_next()
  local str = multitext.selectedtext
  if (str and str:len()~=0) then
    if (not find_dlg) then
      find_dlg = create_find_dialog()  -- create as global, not local
    end

    local find_txt = find_dlg.find_txt
    find_txt.value = str

    item_find_next:action()
  end
end

function item_copy:action()
  local clipboard = iup.clipboard{text = multitext.selectedtext}
  clipboard:destroy()
end

function item_paste:action()
  local clipboard = iup.clipboard{}
  multitext.insert = clipboard.text
  clipboard:destroy()
```

```
  return iup.IGNORE  -- avoid system processing for hot keys, to correctly parse li
end

function item_cut:action()
  local clipboard = iup.clipboard{text = multitext.selectedtext}
  multitext.selectedtext = ""
  clipboard:destroy()
end

function item_delete:action()
  multitext.selectedtext = ""
end

function item_select_all:action()
  iup.SetFocus(multitext)
  multitext.selection = "ALL"
end

function item_font:action()
  local font = multitext.font
  local fontdlg = iup.fontdlg{value = font, parentdialog=iup.GetDialog(self)}

  fontdlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(fontdlg.status) == 1) then
    multitext.font = fontdlg.value
    config:SetVariable("MainWindow", "Font", fontdlg.value)
  end

  fontdlg:destroy()
end

function item_toolbar:action()
  toggle_bar_visibility(self, toolbar_hb)
  config:SetVariable("MainWindow", "Toolbar", item_toolbar.value)
end

function item_statusbar:action()
  toggle_bar_visibility(self, lbl_statusbar)
  config:SetVariable("MainWindow", "Statusbar", item_statusbar.value)
end
```

```
function item_help:action()
  iup.Help("http://www.tecgraf.puc-rio.br/iup")
end

function item_about:action()
  iup.Message("About", "␣␣␣Simple␣Notepad\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣
end


--******************************** Main (Part 2/2) ****************************


btn_new = iup.button{image = "IUP_FileNew", flat = "Yes", action = item_new.action,
btn_open = iup.button{image = "IUP_FileOpen", flat = "Yes", action = item_open.actio
btn_save = iup.button{image = "IUP_FileSave", flat = "Yes", action = item_save.actio
btn_find = iup.button{image = "IUP_EditFind", flat = "Yes", action = item_find.actio
btn_cut = iup.button{image = "IUP_EditCut", flat = "Yes", action = item_cut.action,
btn_copy = iup.button{image =  "IUP_EditCopy", flat = "Yes", action = item_copy.act
btn_paste = iup.button{image = "IUP_EditPaste", flat = "Yes", action = item_paste.a

toolbar_hb = iup.hbox{
  btn_new,
  btn_open,
  btn_save,
  iup.label{separator="VERTICAL"},
  btn_cut,
  btn_copy,
  btn_paste,
  iup.label{separator="VERTICAL"},
  btn_find,
  margin = "5x5",
  gap = 2,
}

vbox = iup.vbox{
  toolbar_hb,
  multitext,
  lbl_statusbar,
}

dlg = iup.dialog{
  vbox,
```

```
  title = "Simple Notepad",
  size = "HALFxHALF",
  menu = menu,
  close_cb = item_exit.action,
  multitext = multitext,
  dropfiles_cb = multitext.dropfiles_cb,
}

function dlg:k_any(c)
  if (c == iup.K_cN) then
    item_new:action()
  elseif (c == iup.K_cO) then
    item_open:action()
  elseif (c == iup.K_cS) then
    item_save:action()
  elseif (c == iup.K_cF) then
    item_find:action()
  elseif (c == iup.K_cH) then
    item_replace:action()
    return iup.IGNORE -- replace system processing
  elseif (c == iup.K_cG) then
    item_goto:action()
  elseif (c == iup.K_F3) then
    item_find_next:action()
  elseif (c == iup.K_cF3) then
    selection_find_next()
  elseif (c == iup.K_cV) then
    item_paste:action()
    return iup.IGNORE -- replace system processing
  end
  -- Ctrl+C, Ctrl+X, Ctrl+A, Del, already implemented inside IupText
end

-- parent for pre-defined dialogs in closed functions (IupMessage and IupAlarm)
iup.SetGlobal("PARENTDIALOG", iup.SetHandleName(dlg))

-- Initialize variables from the configuration file

config:RecentInit(recent_menu, 10)

font = config:GetVariable("MainWindow", "Font")
if (font) then
```

```
  multitext.font = font
end

show_statusbar = config:GetVariableDef("MainWindow", "Statusbar", "ON")
if (show_statusbar == "OFF") then
  item_statusbar.value = "OFF"
  lbl_statusbar.floating = "YES"
  lbl_statusbar.visible = "NO"
end

show_toolbar = config:GetVariableDef("MainWindow", "Toolbar", "ON")
if (show_toolbar == "OFF") then
  item_toolbar.value = "OFF"
  toolbar_hb.floating = "YES"
  toolbar_hb.visible = "NO"
end

-- show the dialog at the last position, with the last size
config:DialogShow(dlg, "MainWindow")

-- initialize the current file
new_file(dlg)

-- open a file from the command line (allow file association in Windows)
if (arg and arg[1]) then
  filename = arg[1]
  open_file(dlg, filename)
end

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

如果我们使用 *IupScintilla* 元素替代 *IupText* 元素，就会发现新大陆。

C语言

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <iup.h>
#include <iup_scintilla.h>
```

```
#include <iup_config.h>


/********************************* Utilities ************************************

void toggleMarker(Ihandle *ih, int lin, int margin)
{
  long int value = IupGetIntId(ih, "MARKERGET", lin);

  if (margin == 1)
    value = value & 0x000001;
  else
    value = value & 0x000002;

  if (value)
    IupSetIntId(ih, "MARKERDELETE", lin, margin - 1);
  else
    IupSetIntId(ih, "MARKERADD", lin, margin - 1);
}

long int setMarkerMask(int markNumber)
{
  long int mask = 0x000000;
  long int mark = 0x00001 << markNumber;
  return mask | mark;
}

void copyMarkedLines(Ihandle *multitext)
{
  int size = IupGetInt(multitext, "COUNT");
  char *buffer = (char *) malloc(size);
  char *text;
  int lin = 0;

  buffer[0] = 0;
  while (lin >= 0)
  {
    IupSetIntId(multitext, "MARKERNEXT", lin, setMarkerMask(0));
    lin = IupGetInt(multitext, "LASTMARKERFOUND");
    if (lin >= 0)
    {
      text = IupGetAttributeId(multitext, "LINE", lin);
```

```
      strcat(buffer, text);  size -= (int)strlen(text);
      lin++;
    }
  }

  if (strlen(buffer) > 0)
  {
    Ihandle *clipboard = IupClipboard();
    IupSetAttribute(clipboard, "TEXT", buffer);
    IupDestroy(clipboard);
  }

  free(buffer);
}

void cutMarkedLines(Ihandle *multitext)
{
  int size = IupGetInt(multitext, "COUNT");
  char *buffer = (char *)malloc(size);
  char *text;
  int lin = 0, pos, len;

  buffer[0] = 0;
  while (lin >= 0 && size)
  {
    IupSetIntId(multitext, "MARKERNEXT", lin, setMarkerMask(0));
    lin = IupGetInt(multitext, "LASTMARKERFOUND");
    if (lin >= 0)
    {
      text = IupGetAttributeId(multitext, "LINE", lin);
      len = (int)strlen(text);
      IupTextConvertLinColToPos(multitext, lin, 0, &pos);
      IupSetStrf(multitext, "DELETERANGE", "%d,%d", pos, len);
      strcat(buffer, text);  size -= len;
      IupSetIntId(multitext, "MARKERDELETE", lin, 0);
      lin--;
    }
  }

  if (strlen(buffer) > 0)
  {
    Ihandle *clipboard = IupClipboard();
```

```
    IupSetAttribute(clipboard, "TEXT", buffer);
    IupDestroy(clipboard);
  }

  free(buffer);
}

void pasteToMarkedLines(Ihandle *multitext)
{
  char *text;
  int lin = 0, pos, len;

  while (lin >= 0)
  {
    IupSetIntId(multitext, "MARKERNEXT", lin, setMarkerMask(0));
    lin = IupGetInt(multitext, "LASTMARKERFOUND");
    if (lin >= 0)
    {
      Ihandle *clipboard;
      text = IupGetAttributeId(multitext, "LINE", lin);
      len = (int)strlen(text);
      IupTextConvertLinColToPos(multitext, lin, 0, &pos);
      IupSetStrf(multitext, "DELETERANGE", "%d,%d", pos, len);
      IupSetIntId(multitext, "MARKERDELETE", lin, 0);
      clipboard = IupClipboard();
      IupSetAttributeId(multitext, "INSERT", pos, IupGetAttribute(clipboard, "TEXT"
      IupDestroy(clipboard);
      lin--;
    }
  }
}

void invertMarkedLines(Ihandle *multitext)
{
  int lin;
  for (lin = 0; lin < IupGetInt(multitext, "LINECOUNT"); lin++)
  {
    toggleMarker(multitext, lin, 1);
  }
}

void removeMarkedLines(Ihandle *multitext)
```

```
{
  char *text;
  int lin = 0, pos, len;

  while (lin >= 0)
  {
    IupSetIntId(multitext, "MARKERNEXT", lin, setMarkerMask(0));
    lin = IupGetInt(multitext, "LASTMARKERFOUND");
    if (lin >= 0)
    {
      text = IupGetAttributeId(multitext, "LINE", lin);
      len = (int)strlen(text);
      IupTextConvertLinColToPos(multitext, lin, 0, &pos);
      IupSetStrf(multitext, "DELETERANGE", "%d,%d", pos, len);
      IupSetIntId(multitext, "MARKERDELETE", lin, 0);
      lin--;
    }
  }
}

void removeUnmarkedLines(Ihandle *multitext)
{
  char *text;
  int len, start = IupGetInt(multitext, "LINECOUNT") - 1, end, posStart, posEnd;

  while (start >= 0)
  {
    text = IupGetAttributeId(multitext, "LINE", start);
    len = (int)strlen(text);
    IupSetIntId(multitext, "MARKERPREVIOUS", start, setMarkerMask(0));
    end = IupGetInt(multitext, "LASTMARKERFOUND");
    IupTextConvertLinColToPos(multitext, start, len + 1, &posEnd);
    if (end >= 0)
    {
      text = IupGetAttributeId(multitext, "LINE", end);
      len = (int)strlen(text);
      IupTextConvertLinColToPos(multitext, end, len + 1, &posStart);
    }
    else
    {
      posStart = 0;
      posEnd++;
```

```
    }
    IupSetStrf(multitext, "DELETERANGE", "%d,%d", posStart, posEnd - posStart);
    end--;
    start = end;
  }
}

void changeTabsToSpaces(Ihandle *multitext)
{
  char *text = IupGetAttribute(multitext, "VALUE");
  int count = IupGetInt(multitext, "COUNT");
  int tabSize = IupGetInt(multitext, "TABSIZE");
  int i, j, lin, col, spacesToNextTab;

  for (i = count - 1; i >= 0; i--)
  {
    char c = text[i];

    if (c != '\t')
      continue;

    IupTextConvertPosToLinCol(multitext, i, &lin, &col);

    spacesToNextTab = tabSize - (col + 1) % tabSize + 1;

    IupSetStrf(multitext, "DELETERANGE", "%d,%d", i, 1);

    for (j = 0; j < spacesToNextTab; j++)
      IupSetAttributeId(multitext, "INSERT", i + j, " ");
  }
}

void changeSpacesToTabs(Ihandle *multitext)
{
  char *text = IupGetAttribute(multitext, "VALUE");
  int count = IupGetInt(multitext, "COUNT");
  int tabSize = IupGetInt(multitext, "TABSIZE");
  int i, lin, col, nSpaces;

  for (i = count - 1; i >= 0; i--)
  {
    char c = text[i];
```

```
    IupTextConvertPosToLinCol(multitext, i, &lin, &col);

    int tabStop = (col + 1) % tabSize == tabSize - 1 ? 1 : 0;

    if (!tabStop || c != ' ')
      continue;

    IupSetStrf(multitext, "DELETERANGE", "%d,%d", i + 1, 1);
    IupSetAttributeId(multitext, "INSERT", i + 1, "\t");

    nSpaces = 0;

    while (text[i - nSpaces] == ' ' && nSpaces < tabSize - 1)
      nSpaces++;

    if (nSpaces == 0)
      continue;

    i -= nSpaces;

    IupSetStrf(multitext, "DELETERANGE", "%d,%d", i + 1, nSpaces);
  }
}

void changeLeadingSpacesToTabs(Ihandle *multitext)
{
  int lineCount = IupGetInt(multitext, "LINECOUNT");
  int tabSize = IupGetInt(multitext, "TABSIZE");
  int i, j, pos, tabCount, spaceCount;

  for (i = 0; i < lineCount; i++)
  {
    char *text = IupGetAttributeId(multitext, "LINE", i);

    int len = (int)strspn(text, " \t");
    if (len == 0)
      continue;

    tabCount = 0;
    spaceCount = 0;
    for (j = 0; j < len; j++)
```

```
    {
      if (text[j] == '\t')
      {
        tabCount++;
        spaceCount = 0;
      }
      else
        spaceCount++;

      if (spaceCount == tabSize)
      {
        tabCount++;
        spaceCount = 0;
      }
    }
    IupTextConvertLinColToPos(multitext, i, 0, &pos);
    IupSetStrf(multitext, "DELETERANGE", "%d,%d", pos, len);
    for (j = 0; j < spaceCount; j++)
      IupSetAttributeId(multitext, "INSERT", pos, " ");
    for (j = 0; j < tabCount; j++)
      IupSetAttributeId(multitext, "INSERT", pos, "\t");
  }
}

void removeLeadingSpaces(Ihandle *multitext)
{
  int lineCount = IupGetInt(multitext, "LINECOUNT");
  int i, pos;

  for (i = 0; i < lineCount; i++)
  {
    char *text = IupGetAttributeId(multitext, "LINE", i);

    int len = (int)strspn(text, " \t");
    if (len == 0)
      continue;

    IupTextConvertLinColToPos(multitext, i, 0, &pos);
    IupSetStrf(multitext, "DELETERANGE", "%d,%d", pos, len);
  }
}
```

```
void removeTrailingSpaces(Ihandle *multitext)
{
  int lineCount = IupGetInt(multitext, "LINECOUNT");
  int i, j, pos, count;

  for (i = 0; i < lineCount; i++)
  {
    char *text = IupGetAttributeId(multitext, "LINE", i);

    int len = (int)strlen(text);
    if (len == 0)
      continue;

    if (text[len - 1] == '\n')
      len--;

    count = 0;
    for (j = len - 1; j >= 0; j--)
    {
      if (text[j] != ' ' && text[j] != '\t')
        break;
      count++;
    }

    if (count == 0)
      continue;

    IupTextConvertLinColToPos(multitext, i, len - count, &pos);
    IupSetStrf(multitext, "DELETERANGE", "%d,%d", pos, count);
  }
}

void changeEolToSpace(Ihandle *multitext)
{
  while (1)
  {
    int pos;
    char *text = IupGetAttribute(multitext, "VALUE");

    char *c = strchr(text, '\n');
    if (c==NULL)
      break;
```

```
    pos = (int)(c - text);

    IupSetStrf(multitext, "DELETERANGE", "%d,%d", pos, 1);
    IupSetAttributeId(multitext, "INSERT", pos, " ");
  }
}

const char* str_filetitle(const char *filename)
{
  /* Start at the last character */
  int len = (int)strlen(filename);
  int offset = len - 1;
  while (offset != 0)
  {
    if (filename[offset] == '\\' || filename[offset] == '/')
    {
      offset++;
      break;
    }
    offset--;
  }
  return filename + offset;
}

char* read_file(const char* filename)
{
  int size;
  char* str;
  FILE* file = fopen(filename, "rb");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return NULL;
  }

  /* calculate file size */
  fseek(file, 0, SEEK_END);
  size = ftell(file);
  fseek(file, 0, SEEK_SET);

  /* allocate memory for the file contents + nul terminator */
```

```
  str = malloc(size + 1);
  /* read all data at once */
  fread(str, size, 1, file);
  /* set the nul terminator */
  str[size] = 0;

  if (ferror(file))
    IupMessagef("Error", "Fail when reading from file: %s", filename);

  fclose(file);
  return str;
}

int write_file(const char* filename, const char* str, int count)
{
  FILE* file = fopen(filename, "w");
  if (!file)
  {
    IupMessagef("Error", "Can't open file: %s", filename);
    return 0;
  }

  fwrite(str, 1, count, file);

  if (ferror(file))
    IupMessagef("Error", "Fail when writing to file: %s", filename);

  fclose(file);
  return 1;
}

void new_file(Ihandle* ih)
{
  Ihandle* dlg = IupGetDialog(ih);
  Ihandle* multitext = IupGetDialogChild(dlg, "MULTITEXT");

  IupSetAttribute(dlg, "TITLE", "Untitled - Scintilla Notepad");
  IupSetAttribute(multitext, "FILENAME", NULL);
  IupSetAttribute(multitext, "DIRTY", "NO");
  IupSetAttribute(multitext, "VALUE", "");
}
```

```
void open_file(Ihandle* ih, const char* filename)
{
  char* str = read_file(filename);
  if (str)
  {
    Ihandle* dlg = IupGetDialog(ih);
    Ihandle* multitext = IupGetDialogChild(dlg, "MULTITEXT");
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

    IupSetfAttribute(dlg, "TITLE", "%s - Scintilla Notepad", str_filetitle(filename
    IupSetStrAttribute(multitext, "FILENAME", filename);
    IupSetAttribute(multitext, "DIRTY", "NO");
    IupSetStrAttribute(multitext, "VALUE", str);

    IupConfigRecentUpdate(config, filename);

    free(str);
  }
}

void save_file(Ihandle* multitext)
{
  char* filename = IupGetAttribute(multitext, "FILENAME");
  char* str = IupGetAttribute(multitext, "VALUE");
  int count = IupGetInt(multitext, "COUNT");
  if (write_file(filename, str, count))
    IupSetAttribute(multitext, "DIRTY", "NO");
}

void saveas_file(Ihandle* multitext, const char* filename)
{
  char* str = IupGetAttribute(multitext, "VALUE");
  int count = IupGetInt(multitext, "COUNT");
  if (write_file(filename, str, count))
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

    IupSetfAttribute(IupGetDialog(multitext), "TITLE", "%s - Scintilla Notepad", st
    IupSetStrAttribute(multitext, "FILENAME", filename);
    IupSetAttribute(multitext, "DIRTY", "NO");

    IupConfigRecentUpdate(config, filename);
```

```
  }
}

int save_check(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");
  if (IupGetInt(multitext, "DIRTY"))
  {
    switch (IupAlarm("Warning", "File not saved! Save it now?", "Yes", "No", "Cance
    {
    case 1:  /* save the changes and continue */
      save_file(multitext);
      break;
    case 2:  /* ignore the changes and continue */
      break;
    case 3:  /* cancel */
      return 0;
    }
  }
  return 1;
}

void toggle_bar_visibility(Ihandle* item, Ihandle* ih)
{
  if (IupGetInt(item, "VALUE"))
  {
    IupSetAttribute(ih, "FLOATING", "YES");
    IupSetAttribute(ih, "VISIBLE", "NO");
    IupSetAttribute(item, "VALUE", "OFF");
  }
  else
  {
    IupSetAttribute(ih, "FLOATING", "NO");
    IupSetAttribute(ih, "VISIBLE", "YES");
    IupSetAttribute(item, "VALUE", "ON");
  }

  IupRefresh(ih);  /* refresh the dialog layout */
}

void set_find_replace_visibility(Ihandle* find_dlg, int show_replace)
{
```

```
  Ihandle* replace_txt = IupGetDialogChild(find_dlg, "REPLACE_TEXT");
  Ihandle* replace_lbl = IupGetDialogChild(find_dlg, "REPLACE_LABEL");
  Ihandle* replace_bt = IupGetDialogChild(find_dlg, "REPLACE_BUTTON");

  if (show_replace)
  {
    IupSetAttribute(replace_txt, "VISIBLE", "Yes");
    IupSetAttribute(replace_lbl, "VISIBLE", "Yes");
    IupSetAttribute(replace_bt, "VISIBLE", "Yes");
    IupSetAttribute(replace_txt, "FLOATING", "No");
    IupSetAttribute(replace_lbl, "FLOATING", "No");
    IupSetAttribute(replace_bt, "FLOATING", "No");

    IupSetAttribute(find_dlg, "TITLE", "Replace");
  }
  else
  {
    IupSetAttribute(replace_txt, "FLOATING", "Yes");
    IupSetAttribute(replace_lbl, "FLOATING", "Yes");
    IupSetAttribute(replace_bt, "FLOATING", "Yes");
    IupSetAttribute(replace_txt, "VISIBLE", "No");
    IupSetAttribute(replace_lbl, "VISIBLE", "No");
    IupSetAttribute(replace_bt, "VISIBLE", "No");

    IupSetAttribute(find_dlg, "TITLE", "Find");
  }

  IupSetAttribute(find_dlg, "SIZE", NULL);  /* force a dialog resize on the IupRefr
  IupRefresh(find_dlg);
}


/********************************* Callbacks ************************************


int dropfiles_cb(Ihandle* ih, const char* filename)
{
  if (save_check(ih))
    open_file(ih, filename);

  return IUP_DEFAULT;
}
```

```
int marginclick_cb(Ihandle* ih, int margin, int lin, char *status)
{
  (void)status;

  if (margin < 1 || margin > 2)
    return IUP_IGNORE;

  toggleMarker(ih, lin, margin);

  return IUP_DEFAULT;
}

int multitext_valuechanged_cb(Ihandle* multitext)
{
  IupSetAttribute(multitext, "DIRTY", "YES");
  return IUP_DEFAULT;
}

int file_menu_open_cb(Ihandle* ih)
{
  Ihandle* item_revert = IupGetDialogChild(ih, "ITEM_REVERT");
  Ihandle* item_save = IupGetDialogChild(ih, "ITEM_SAVE");
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");
  char* filename = IupGetAttribute(multitext, "FILENAME");
  int dirty = IupGetInt(multitext, "DIRTY");

  if (dirty)
    IupSetAttribute(item_save, "ACTIVE", "YES");
  else
    IupSetAttribute(item_save, "ACTIVE", "NO");

  if (dirty && filename)
    IupSetAttribute(item_revert, "ACTIVE", "YES");
  else
    IupSetAttribute(item_revert, "ACTIVE", "NO");
  return IUP_DEFAULT;
}

int edit_menu_open_cb(Ihandle* ih)
{
  Ihandle *clipboard = IupClipboard();
```

```
  Ihandle* find_dlg = (Ihandle*)IupGetAttribute(ih, "FIND_DIALOG");

  Ihandle *item_paste = IupGetDialogChild(ih, "ITEM_PASTE");
  Ihandle *item_cut = IupGetDialogChild(ih, "ITEM_CUT");
  Ihandle *item_delete = IupGetDialogChild(ih, "ITEM_DELETE");
  Ihandle *item_copy = IupGetDialogChild(ih, "ITEM_COPY");
  Ihandle *item_find_next = IupGetDialogChild(ih, "ITEM_FINDNEXT");
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  if (!IupGetInt(clipboard, "TEXTAVAILABLE"))
    IupSetAttribute(item_paste, "ACTIVE", "NO");
  else
    IupSetAttribute(item_paste, "ACTIVE", "YES");

  if (!IupGetAttribute(multitext, "SELECTEDTEXT"))
  {
    IupSetAttribute(item_cut, "ACTIVE", "NO");
    IupSetAttribute(item_delete, "ACTIVE", "NO");
    IupSetAttribute(item_copy, "ACTIVE", "NO");
  }
  else
  {
    IupSetAttribute(item_cut, "ACTIVE", "YES");
    IupSetAttribute(item_delete, "ACTIVE", "YES");
    IupSetAttribute(item_copy, "ACTIVE", "YES");
  }

  if (find_dlg)
  {
    Ihandle* txt = IupGetDialogChild(find_dlg, "FIND_TEXT");
    char* str_to_find = IupGetAttribute(txt, "VALUE");

    if (!str_to_find || str_to_find[0] == 0)
      IupSetAttribute(item_find_next, "ACTIVE", "NO");
    else
      IupSetAttribute(item_find_next, "ACTIVE", "Yes");
  }
  else
    IupSetAttribute(item_find_next, "ACTIVE", "NO");

  IupDestroy(clipboard);
  return IUP_DEFAULT;
```

```
}

int config_recent_cb(Ihandle* ih)
{
  if (save_check(ih))
  {
    char* filename = IupGetAttribute(ih, "TITLE");
    open_file(ih, filename);
  }
  return IUP_DEFAULT;
}

int multitext_caret_cb(Ihandle *ih, int lin, int col)
{
  Ihandle *lbl_statusbar = IupGetDialogChild(ih, "STATUSBAR");
  IupSetfAttribute(lbl_statusbar, "TITLE", "Lin %d, Col %d", lin, col);
  return IUP_DEFAULT;
}

int item_new_action_cb(Ihandle* item_new)
{
  if (save_check(item_new))
    new_file(item_new);

  return IUP_DEFAULT;
}

int item_open_action_cb(Ihandle* item_open)
{
  Ihandle *filedlg;
  Ihandle* config;
  const char* dir;

  if (!save_check(item_open))
    return IUP_DEFAULT;

  config = (Ihandle*)IupGetAttribute(item_open, "CONFIG");
  dir = IupConfigGetVariableStr(config, "MainWindow", "LastDirectory");

  filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "OPEN");
  IupSetAttribute(filedlg, "EXTFILTER", "Text Files|*.txt|All Files|*.*|");
```

```
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_open));
  IupSetStrAttribute(filedlg, "DIRECTORY", dir);

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    open_file(item_open, filename);

    dir = IupGetAttribute(filedlg, "DIRECTORY");
    IupConfigSetVariableStr(config, "MainWindow", "LastDirectory", dir);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_saveas_action_cb(Ihandle* item_saveas)
{
  Ihandle* multitext = IupGetDialogChild(item_saveas, "MULTITEXT");
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
  const char* dir = IupConfigGetVariableStr(config, "MainWindow", "LastDirectory");
  Ihandle *filedlg = IupFileDlg();
  IupSetAttribute(filedlg, "DIALOGTYPE", "SAVE");
  IupSetAttribute(filedlg, "EXTFILTER", "Text␣Files|*.txt|All␣Files|*.*|");
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", IupGetDialog(item_saveas));
  IupSetStrAttribute(filedlg, "FILE", IupGetAttribute(multitext, "FILENAME"));
  IupSetStrAttribute(filedlg, "DIRECTORY", dir);

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    saveas_file(multitext, filename);

    dir = IupGetAttribute(filedlg, "DIRECTORY");
    IupConfigSetVariableStr(config, "MainWindow", "LastDirectory", dir);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
```

```
}

int item_save_action_cb(Ihandle* item_save)
{
  Ihandle* multitext = IupGetDialogChild(item_save, "MULTITEXT");
  char* filename = IupGetAttribute(multitext, "FILENAME");
  if (!filename)
    item_saveas_action_cb(item_save);
  else
  {
    /* test again because in can be called using the hot key */
    int dirty = IupGetInt(multitext, "DIRTY");
    if (dirty)
      save_file(multitext);
  }
  return IUP_DEFAULT;
}

int item_revert_action_cb(Ihandle* item_revert)
{
  Ihandle* multitext = IupGetDialogChild(item_revert, "MULTITEXT");
  char* filename = IupGetAttribute(multitext, "FILENAME");
  open_file(item_revert, filename);
  return IUP_DEFAULT;
}

int item_exit_action_cb(Ihandle* item_exit)
{
  Ihandle* dlg = IupGetDialog(item_exit);
  Ihandle* config = (Ihandle*)IupGetAttribute(dlg, "CONFIG");

  if (!save_check(item_exit))
    return IUP_IGNORE;  /* to abort the CLOSE_CB callback */

  IupConfigDialogClosed(config, dlg, "MainWindow");
  IupConfigSave(config);
  IupDestroy(config);
  return IUP_CLOSE;
}

int goto_ok_action_cb(Ihandle* bt_ok)
{
```

```
  int line_count = IupGetInt(bt_ok, "TEXT_LINECOUNT");
  Ihandle* txt = IupGetDialogChild(bt_ok, "LINE_TEXT");
  int line = IupGetInt(txt, "VALUE");
  if (line < 1 || line >= line_count)
  {
    IupMessage("Error", "Invalid line number.");
    return IUP_DEFAULT;
  }

  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "1");
  return IUP_CLOSE;
}

int goto_cancel_action_cb(Ihandle* bt_ok)
{
  IupSetAttribute(IupGetDialog(bt_ok), "STATUS", "0");
  return IUP_CLOSE;
}

int item_goto_action_cb(Ihandle* item_goto)
{
  Ihandle* multitext = IupGetDialogChild(item_goto, "MULTITEXT");
  Ihandle *goto_dlg, *box, *bt_ok, *bt_cancel, *txt, *lbl;
  int line_count = IupGetInt(multitext, "LINECOUNT");

  lbl = IupLabel(NULL);
  IupSetfAttribute(lbl, "TITLE", "Line Number [1-%d]:", line_count);
  txt = IupText(NULL);
  IupSetAttribute(txt, "MASK", IUP_MASK_UINT);  /* unsigned integer numbers only */
  IupSetAttribute(txt, "NAME", "LINE_TEXT");
  IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
  bt_ok = IupButton("OK", NULL);
  IupSetInt(bt_ok, "TEXT_LINECOUNT", line_count);
  IupSetAttribute(bt_ok, "PADDING", "10x2");
  IupSetCallback(bt_ok, "ACTION", (Icallback)goto_ok_action_cb);
  bt_cancel = IupButton("Cancel", NULL);
  IupSetCallback(bt_cancel, "ACTION", (Icallback)goto_cancel_action_cb);
  IupSetAttribute(bt_cancel, "PADDING", "10x2");

  box = IupVbox(
    lbl,
    txt,
```

```
    IupSetAttributes(IupHbox(
    IupFill(),
    bt_ok,
    bt_cancel,
    NULL), "NORMALIZESIZE=HORIZONTAL"),
    NULL);
  IupSetAttribute(box, "MARGIN", "10x10");
  IupSetAttribute(box, "GAP", "5");

  goto_dlg = IupDialog(box);
  IupSetAttribute(goto_dlg, "TITLE", "Go To Line");
  IupSetAttribute(goto_dlg, "DIALOGFRAME", "Yes");
  IupSetAttributeHandle(goto_dlg, "DEFAULTENTER", bt_ok);
  IupSetAttributeHandle(goto_dlg, "DEFAULTESC", bt_cancel);
  IupSetAttributeHandle(goto_dlg, "PARENTDIALOG", IupGetDialog(item_goto));

  IupPopup(goto_dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(goto_dlg, "STATUS") == 1)
  {
    int line = IupGetInt(txt, "VALUE");
    int pos;
    IupTextConvertLinColToPos(multitext, line, 0, &pos);
    IupSetInt(multitext, "CARETPOS", pos);
    IupSetInt(multitext, "SCROLLTOPOS", pos);
  }

  IupDestroy(goto_dlg);

  return IUP_DEFAULT;
}

int item_gotombrace_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  int pos = IupGetInt(multitext, "CARETPOS");

  int newpos = IupGetIntId(multitext, "BRACEMATCH", pos);

  if (newpos != -1)
  {
```

```
    IupSetStrf(multitext, "BRACEHIGHLIGHT", "%d:%d", pos, newpos);

    IupSetInt(multitext, "CARETPOS", newpos);
    IupSetInt(multitext, "SCROLLTOPOS", newpos);
  }

  return IUP_IGNORE;
}

int item_togglemark_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  int pos = IupGetInt(multitext, "CARETPOS");

  int lin, col;
  IupTextConvertPosToLinCol(multitext, pos, &lin, &col);

  toggleMarker(multitext, lin, 1);

  return IUP_IGNORE;
}

int item_nextmark_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  int pos = IupGetInt(multitext, "CARETPOS");

  int lin, col;
  IupTextConvertPosToLinCol(multitext, pos, &lin, &col);

  IupSetIntId(multitext, "MARKERNEXT", lin + 1, setMarkerMask(0));

  lin = IupGetInt(multitext, "LASTMARKERFOUND");

  if (lin == -1)
    return IUP_IGNORE;

  IupTextConvertLinColToPos(multitext, lin, 0, &pos);

  IupSetInt(multitext, "CARETPOS", pos);
```

```
  return IUP_DEFAULT;
}

int item_previousmark_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  int pos = IupGetInt(multitext, "CARETPOS");

  int lin, col;
  IupTextConvertPosToLinCol(multitext, pos, &lin, &col);

  IupSetIntId(multitext, "MARKERPREVIOUS", lin - 1, setMarkerMask(0));

  lin = IupGetInt(multitext, "LASTMARKERFOUND");

  if (lin == -1)
    return IUP_IGNORE;

  IupTextConvertLinColToPos(multitext, lin, 0, &pos);

  IupSetInt(multitext, "CARETPOS", pos);

  return IUP_DEFAULT;
}

int item_clearmarks_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");
  IupSetInt(multitext, "MARKERDELETEALL", 0);
  return IUP_DEFAULT;
}

int item_copymarked_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  copyMarkedLines(multitext);

  return IUP_DEFAULT;
}
```

```
int item_cutmarked_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  cutMarkedLines(multitext);

  return IUP_DEFAULT;
}

int item_pastetomarked_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  pasteToMarkedLines(multitext);

  return IUP_DEFAULT;
}

int item_removemarked_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  removeMarkedLines(multitext);

  return IUP_DEFAULT;
}

int item_removeunmarked_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  removeUnmarkedLines(multitext);

  return IUP_DEFAULT;
}

int item_invertmarks_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  invertMarkedLines(multitext);
```

```
  return IUP_DEFAULT;
}

int item_eoltospace_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  IupSetAttribute(multitext, "UNDOACTION", "BEGIN");

  changeEolToSpace(multitext);

  IupSetAttribute(multitext, "UNDOACTION", "END");

  return IUP_DEFAULT;
}

int item_removespaceeol_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  IupSetAttribute(multitext, "UNDOACTION", "BEGIN");

  removeTrailingSpaces(multitext);

  removeLeadingSpaces(multitext);

  changeEolToSpace(multitext);

  IupSetAttribute(multitext, "UNDOACTION", "END");

  return IUP_DEFAULT;
}

int item_trimtrailing_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  IupSetAttribute(multitext, "UNDOACTION", "BEGIN");

  removeTrailingSpaces(multitext);
```

```
  IupSetAttribute(multitext, "UNDOACTION", "END");

  return IUP_DEFAULT;
}

int item_trimleading_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  IupSetAttribute(multitext, "UNDOACTION", "BEGIN");

  removeLeadingSpaces(multitext);

  IupSetAttribute(multitext, "UNDOACTION", "END");

  return IUP_DEFAULT;
}

int item_trimtraillead_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  IupSetAttribute(multitext, "UNDOACTION", "BEGIN");

  removeTrailingSpaces(multitext);

  removeLeadingSpaces(multitext);

  IupSetAttribute(multitext, "UNDOACTION", "END");

  return IUP_DEFAULT;
}

int item_tabtospace_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  IupSetAttribute(multitext, "UNDOACTION", "BEGIN");

  changeTabsToSpaces(multitext);

  IupSetAttribute(multitext, "UNDOACTION", "END");
```

```
  return IUP_DEFAULT;
}

int item_allspacetotab_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  IupSetAttribute(multitext, "UNDOACTION", "BEGIN");

  changeTabsToSpaces(multitext);

  changeSpacesToTabs(multitext);

  IupSetAttribute(multitext, "UNDOACTION", "END");

  return IUP_DEFAULT;
}

int item_leadingspacetotab_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  IupSetAttribute(multitext, "UNDOACTION", "BEGIN");

  changeLeadingSpacesToTabs(multitext);

  IupSetAttribute(multitext, "UNDOACTION", "END");

  return IUP_DEFAULT;
}

int find_next_action_cb(Ihandle* ih)
{
  /* this callback can be called from the main dialog also */
  Ihandle* find_dlg = (Ihandle*)IupGetAttribute(ih, "FIND_DIALOG");
  if (find_dlg)
  {
    char* str;
    int pos;
    Ihandle* multitext = (Ihandle*)IupGetAttribute(find_dlg, "MULTITEXT");
    str = IupGetAttribute(multitext, "VALUE");
```

```
Ihandle* txt = IupGetDialogChild(find_dlg, "FIND_TEXT");
char* str_to_find = IupGetAttribute(txt, "VALUE");

Ihandle* wrapHnd = IupGetDialogChild(find_dlg, "WRAP");
int wrap = IupGetInt(wrapHnd, "VALUE");

Ihandle* downHnd = IupGetDialogChild(find_dlg, "DOWN");
int down = IupGetInt(downHnd, "VALUE");

Ihandle* findHnd = IupGetDialogChild(find_dlg, "FIND_CASE");
int casesensitive = IupGetInt(findHnd, "VALUE");
Ihandle* wholeHnd = IupGetDialogChild(find_dlg, "WHOLE_WORD");
int whole_word = IupGetInt(wholeHnd, "VALUE");
Ihandle* wordHnd = IupGetDialogChild(find_dlg, "WORD_START");
int word_start = IupGetInt(wordHnd, "VALUE");
Ihandle* regexpHnd = IupGetDialogChild(find_dlg, "REGEXP");
int regexp = IupGetInt(regexpHnd, "VALUE");
Ihandle* posixHnd = IupGetDialogChild(find_dlg, "POSIX");
int posix = IupGetInt(posixHnd, "VALUE");

char flags[80];
flags[0] = 0;
IupSetAttribute(multitext, "SEARCHFLAGS", NULL);
if (casesensitive)
  strcpy(flags, "MATCHCASE");
if (whole_word)
  strcat((flags[0] != 0 ? strcat(flags, "␣|␣") : flags), "WHOLEWORD");
if (word_start)
  strcat((flags[0] != 0 ? strcat(flags, "␣|␣") : flags), "WORDSTART");
if (regexp)
  strcat((flags[0] != 0 ? strcat(flags, "␣|␣") : flags), "REGEXP");
if (posix)
  strcat((flags[0] != 0 ? strcat(flags, "␣|␣") : flags), "POSIX");

if (flags[0] != 0)
  IupSetAttribute(multitext, "SEARCHFLAGS", flags);

/* test again, because it can be called from the hot key */
if (!str_to_find || str_to_find[0] == 0)
  return IUP_DEFAULT;
```

```
  char *sel;
  int find_pos = IupGetInt(multitext, "CARETPOS");
  sel = IupGetAttribute(multitext, "SELECTIONPOS");

  if (!down && sel)
  {
    int st, ed;
    IupGetIntInt(multitext, "SELECTIONPOS", &st, &ed);
    find_pos = st;
  }

  IupSetInt(multitext, "TARGETSTART", find_pos);
  IupSetInt(multitext, "TARGETEND", down ? IupGetInt(multitext, "COUNT") - 1 : 1)

  IupSetAttribute(multitext, "SEARCHINTARGET", str_to_find);

  pos = IupGetInt(multitext, "TARGETSTART");

  if (pos == find_pos && wrap)
  {
    IupSetInt(multitext, "TARGETSTART", down ? 1 : IupGetInt(multitext, "COUNT")
    IupSetInt(multitext, "TARGETEND", find_pos);

    IupSetAttribute(multitext, "SEARCHINTARGET", str_to_find);

    pos = IupGetInt(multitext, "TARGETSTART");
  }

  if (pos != find_pos)
  {
    int lin, col, end_pos, start_pos;

    start_pos = IupGetInt(multitext, "TARGETSTART");
    end_pos = IupGetInt(multitext, "TARGETEND");

    IupSetFocus(multitext);
    IupSetfAttribute(multitext, "SELECTIONPOS", "%d:%d", pos, end_pos);

    //IupSetInt(multitext, "TARGETSTART", down ? end_pos : start_pos);
    //IupSetInt(multitext, "TARGETEND", down ? IupGetInt(multitext, "COUNT") - 1

    IupTextConvertPosToLinCol(multitext, end_pos, &lin, &col);
```

```
      multitext_caret_cb(multitext, lin, col);
    }
    else
      IupMessage("Warning", "Text not found.");
  }

  return IUP_DEFAULT;
}

int find_replace_action_cb(Ihandle* bt_replace)
{
  Ihandle* find_dlg = (Ihandle*)IupGetAttribute(bt_replace, "FIND_DIALOG");
  Ihandle* multitext = (Ihandle*)IupGetAttribute(find_dlg, "MULTITEXT");
  char* selectionpos = IupGetAttribute(multitext, "SELECTIONPOS");

  if (!selectionpos)
    find_next_action_cb(bt_replace);
  else
  {
    Ihandle* replace_txt = IupGetDialogChild(find_dlg, "REPLACE_TEXT");
    char* str_to_replace = IupGetAttribute(replace_txt, "VALUE");
    if (IupGetAttribute(multitext, "SELECTIONPOS"))
    {
      int init, fim;
      IupGetIntInt(multitext, "SELECTIONPOS", &init, &fim);
      IupSetInt(multitext, "TARGETSTART", init);
      IupSetInt(multitext, "TARGETEND", fim);
      IupSetAttribute(multitext, "REPLACETARGET", str_to_replace);
    }

    /* then find next */
    find_next_action_cb(bt_replace);
  }

  return IUP_DEFAULT;
}

int find_close_action_cb(Ihandle* bt_close)
{
  Ihandle* find_dlg = IupGetDialog(bt_close);
  Ihandle* multitext = (Ihandle*)IupGetAttribute(find_dlg, "MULTITEXT");
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
```

```
  IupConfigDialogClosed(config, find_dlg, "FindDialog");
  IupHide(find_dlg);  /* do not destroy, just hide */
  return IUP_DEFAULT;
}

Ihandle* create_find_dialog(Ihandle *multitext)
{
  Ihandle *box, *bt_next, *bt_close, *txt, *find_dlg;
  Ihandle *find_case, *whole_word, *word_start, *reg_exp, *posix, *wrap, *up, *down
  Ihandle *flags, *direction, *radio;
  Ihandle *txt_replace, *bt_replace;

  txt = IupText(NULL);
  IupSetAttribute(txt, "NAME", "FIND_TEXT");
  IupSetAttribute(txt, "VISIBLECOLUMNS", "20");
  txt_replace = IupText(NULL);
  IupSetAttribute(txt_replace, "NAME", "REPLACE_TEXT");
  IupSetAttribute(txt_replace, "VISIBLECOLUMNS", "20");
  find_case = IupToggle("Case Sensitive", NULL);
  IupSetAttribute(find_case, "NAME", "FIND_CASE");
  whole_word = IupToggle("Whole Word", NULL);
  IupSetAttribute(whole_word, "NAME", "WHOLE_WORD");
  word_start = IupToggle("Word Start", NULL);
  IupSetAttribute(word_start, "NAME", "WORD_START");
  reg_exp = IupToggle("Regular Expression", NULL);
  IupSetAttribute(reg_exp, "NAME", "REG_EXP");
  posix = IupToggle("Posix", NULL);
  IupSetAttribute(posix, "NAME", "POSIX");
  wrap = IupToggle("Wrap Around", NULL);
  IupSetAttribute(wrap, "NAME", "WRAP");
  up = IupToggle("Up", NULL);
  IupSetAttribute(up, "NAME", "UP");
  down = IupToggle("Down", NULL);
  IupSetAttribute(down, "NAME", "DOWN");
  bt_next = IupButton("Find Next", NULL);
  IupSetAttribute(bt_next, "PADDING", "10x2");
  IupSetCallback(bt_next, "ACTION", (Icallback)find_next_action_cb);
  bt_replace = IupButton("Replace", NULL);
  IupSetAttribute(bt_replace, "PADDING", "10x2");
  IupSetCallback(bt_replace, "ACTION", (Icallback)find_replace_action_cb);
  IupSetAttribute(bt_replace, "NAME", "REPLACE_BUTTON");
  bt_close = IupButton("Close", NULL);
```

```
  IupSetCallback(bt_close, "ACTION", (Icallback)find_close_action_cb);
  IupSetAttribute(bt_close, "PADDING", "10x2");

  flags = IupVbox(find_case,
                  whole_word,
                  word_start,
                  reg_exp,
                  posix,
                  wrap,
                  NULL);

  radio = IupRadio(IupVbox(up,
    down,
    NULL));

  IupSetAttribute(radio, "VALUE_HANDLE", (char*)down);

  direction = IupFrame(radio);

  IupSetAttribute(direction, "TITLE", "Direction");

  box = IupVbox(
    IupLabel("Find␣What:"),
    txt,
    IupSetAttributes(IupLabel("Replace␣with:"), "NAME=REPLACE_LABEL"),
    txt_replace,
    IupHbox(
    flags,
    direction,
    NULL),
    IupSetAttributes(IupHbox(
    IupFill(),
    bt_next,
    bt_replace,
    bt_close,
    NULL), "NORMALIZESIZE=HORIZONTAL"),
    NULL);
  IupSetAttribute(box, "NMARGIN", "10x10");
  IupSetAttribute(box, "GAP", "5");

  find_dlg = IupDialog(box);
  IupSetAttribute(find_dlg, "TITLE", "Find");
```

```
  IupSetAttribute(find_dlg, "DIALOGFRAME", "Yes");
  IupSetAttributeHandle(find_dlg, "DEFAULTENTER", bt_next);
  IupSetAttributeHandle(find_dlg, "DEFAULTESC", bt_close);
  IupSetAttributeHandle(find_dlg, "PARENTDIALOG", IupGetDialog(multitext));
  IupSetCallback(find_dlg, "CLOSE_CB", (Icallback)find_close_action_cb);

  /* Save the multiline to access it from the callbacks */
  IupSetAttribute(find_dlg, "MULTITEXT", (char*)multitext);

  IupSetInt(multitext, "TARGETSTART", IupGetInt(multitext, "CARETPOS"));
  IupSetInt(multitext, "TARGETEND", IupGetInt(multitext, "COUNT") - 1);

  /* Save the dialog to reuse it */
  IupSetAttribute(find_dlg, "FIND_DIALOG", (char*)find_dlg);  /* from itself */
  IupSetAttribute(IupGetDialog(multitext), "FIND_DIALOG", (char*)find_dlg); /* from

  return find_dlg;
}

int item_find_action_cb(Ihandle* item_find)
{
  Ihandle* find_dlg = (Ihandle*)IupGetAttribute(item_find, "FIND_DIALOG");
  Ihandle* multitext = IupGetDialogChild(item_find, "MULTITEXT");
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
  char* str;

  if (!find_dlg)
    find_dlg = create_find_dialog(multitext);

  set_find_replace_visibility(find_dlg, 0);

  IupConfigDialogShow(config, find_dlg, "FindDialog");

  str = IupGetAttribute(multitext, "SELECTEDTEXT");
  if (str && str[0] != 0)
  {
    Ihandle* txt = IupGetDialogChild(find_dlg, "FIND_TEXT");
    IupSetStrAttribute(txt, "VALUE", str);
  }

  return IUP_DEFAULT;
}
```

```
int item_replace_action_cb(Ihandle* item_replace)
{
  Ihandle* find_dlg = (Ihandle*)IupGetAttribute(item_replace, "FIND_DIALOG");
  Ihandle* multitext = IupGetDialogChild(item_replace, "MULTITEXT");
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
  char* str;

  if (!find_dlg)
    find_dlg = create_find_dialog(multitext);

  set_find_replace_visibility(find_dlg, 1);

  IupConfigDialogShow(config, find_dlg, "FindDialog");

  str = IupGetAttribute(multitext, "SELECTEDTEXT");
  if (str && str[0] != 0)
  {
    Ihandle* txt = IupGetDialogChild(find_dlg, "FIND_TEXT");
    IupSetStrAttribute(txt, "VALUE", str);
  }

  return IUP_IGNORE;  /* replace system processing for the hot key */
}

int selection_find_next_action_cb(Ihandle* ih)
{
  Ihandle* multitext = IupGetDialogChild(ih, "MULTITEXT");

  char* str = IupGetAttribute(multitext, "SELECTEDTEXT");
  if (str && str[0] != 0)
  {
    Ihandle* txt;
    Ihandle* find_dlg = (Ihandle*)IupGetAttribute(ih, "FIND_DIALOG");

    if (!find_dlg)
      find_dlg = create_find_dialog(multitext);

    txt = IupGetDialogChild(find_dlg, "FIND_TEXT");
    IupSetStrAttribute(txt, "VALUE", str);

    find_next_action_cb(ih);
```

```
  }

  return IUP_DEFAULT;
}

int item_copy_action_cb(Ihandle* item_copy)
{
  Ihandle* multitext = IupGetDialogChild(item_copy, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(clipboard, "TEXT", IupGetAttribute(multitext, "SELECTEDTEXT"));
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_paste_action_cb(Ihandle* item_paste)
{
  Ihandle* multitext = IupGetDialogChild(item_paste, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(multitext, "INSERT", IupGetAttribute(clipboard, "TEXT"));
  IupDestroy(clipboard);
  return IUP_IGNORE;  /* replace system processing for the hot key, to correctly pa
}

int item_cut_action_cb(Ihandle* item_cut)
{
  Ihandle* multitext = IupGetDialogChild(item_cut, "MULTITEXT");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(clipboard, "TEXT", IupGetAttribute(multitext, "SELECTEDTEXT"));
  IupSetAttribute(multitext, "SELECTEDTEXT", "");
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_delete_action_cb(Ihandle* item_delete)
{
  Ihandle* multitext = IupGetDialogChild(item_delete, "MULTITEXT");
  IupSetAttribute(multitext, "SELECTEDTEXT", "");
  return IUP_DEFAULT;
}

int item_select_all_action_cb(Ihandle* item_select_all)
{
```

```
  Ihandle* multitext = IupGetDialogChild(item_select_all, "MULTITEXT");
  IupSetFocus(multitext);
  int count = IupGetInt(multitext, "COUNT");
  IupSetStrf(multitext, "SELECTIONPOS", "%d:%d", 0, count - 1);
  return IUP_DEFAULT;
}

int item_undo_action_cb(Ihandle* item_select_all)
{
  Ihandle* multitext = IupGetDialogChild(item_select_all, "MULTITEXT");
  IupSetAttribute(multitext, "UNDO", "YES");
  return IUP_DEFAULT;
}

int item_redo_action_cb(Ihandle* item_select_all)
{
  Ihandle* multitext = IupGetDialogChild(item_select_all, "MULTITEXT");
  IupSetAttribute(multitext, "REDO", "YES");
  return IUP_DEFAULT;
}

void StrUpper(char* sstr)
{
  if (!sstr || sstr[0] == 0) return;
  for (; *sstr; sstr++)
    *sstr = (char)toupper(*sstr);
}

void StrLower(char* sstr)
{
  if (!sstr || sstr[0] == 0) return;
  for (; *sstr; sstr++)
    *sstr = (char)tolower(*sstr);
}

char* StrDup(const char *str)
{
  if (str)
  {
    int size = (int)strlen(str) + 1;
    char *newstr = malloc(size);
    if (newstr) memcpy(newstr, str, size);
```

```
    return newstr;
  }
  return NULL;
}

int item_uppercase_action_cb(Ihandle* item)
{
  char *text;
  int start, end;
  Ihandle* multitext = IupGetDialogChild(item, "MULTITEXT");
  IupGetIntInt(multitext, "SELECTIONPOS", &start, &end);
  text = IupGetAttribute(multitext, "SELECTEDTEXT");
  text = StrDup(text);
  StrUpper(text);
  IupSetAttribute(multitext, "SELECTEDTEXT", text);
  IupSetStrf(multitext, "SELECTIONPOS", "%d:%d", start, end);
  free(text);
  return IUP_DEFAULT;
}

int item_lowercase_action_cb(Ihandle* item)
{
  char *text;
  int start, end;
  Ihandle* multitext = IupGetDialogChild(item, "MULTITEXT");
  IupGetIntInt(multitext, "SELECTIONPOS", &start, &end);
  text = IupGetAttribute(multitext, "SELECTEDTEXT");
  text = StrDup(text);
  StrLower(text);
  IupSetAttribute(multitext, "SELECTEDTEXT", text);
  IupSetStrf(multitext, "SELECTIONPOS", "%d:%d", start, end);
  free(text);
  return IUP_DEFAULT;
}

int item_case_action_cb(Ihandle* item)
{
  char* shift = IupGetGlobal("SHIFTKEY");

  if (strcmp(shift, "ON") == 0)
    item_uppercase_action_cb(item);
  else
```

```
    item_lowercase_action_cb(item);

  return IUP_DEFAULT;
}

int item_font_action_cb(Ihandle* item_font)
{
  Ihandle* multitext = IupGetDialogChild(item_font, "MULTITEXT");
  Ihandle* fontdlg = IupFontDlg();
  char* font = IupGetAttribute(multitext, "FONT");
  IupSetStrAttribute(fontdlg, "VALUE", font);
  IupSetAttributeHandle(fontdlg, "PARENTDIALOG", IupGetDialog(item_font));

  IupPopup(fontdlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(fontdlg, "STATUS") == 1)
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");
    font = IupGetAttribute(fontdlg, "VALUE");
    IupSetStrAttribute(multitext, "FONT", font);

    IupConfigSetVariableStr(config, "MainWindow", "Font", font);
  }

  IupDestroy(fontdlg);
  return IUP_DEFAULT;
}

int item_tab_action_cb(Ihandle* item_font)
{
  Ihandle* multitext = IupGetDialogChild(item_font, "MULTITEXT");

  int replaceBySpace = !IupGetInt(multitext, "USETABS");
  int tabSize = IupGetInt(multitext, "TABSIZE");

  if (!IupGetParam("Tab Settings", NULL,
    0,
    "Size: %i\n"
    "Replace by Whitespace: %b\n", &tabSize, &replaceBySpace))
    return IUP_IGNORE;

  IupSetInt(multitext, "TABSIZE", tabSize);
```

```
  IupSetInt(multitext, "USETABS", !replaceBySpace);

  return IUP_DEFAULT;
}

int item_zoomin_action_cb(Ihandle* item_toolbar)
{
  Ihandle* multitext = IupGetDialogChild(item_toolbar, "MULTITEXT");

  IupSetAttribute(multitext, "ZOOMIN", "10");

  return IUP_DEFAULT;
}

int item_zoomout_action_cb(Ihandle* item_toolbar)
{
  Ihandle* multitext = IupGetDialogChild(item_toolbar, "MULTITEXT");

  IupSetAttribute(multitext, "ZOOMOUT", "10");

  return IUP_DEFAULT;
}

int item_restorezoom_action_cb(Ihandle* item_toolbar)
{
  Ihandle* multitext = IupGetDialogChild(item_toolbar, "MULTITEXT");

  IupSetAttribute(multitext, "ZOOM", "0");

  return IUP_DEFAULT;
}

int item_wordwrap_action_cb(Ihandle* item_wordwrap)
{
  Ihandle* multitext = IupGetDialogChild(item_wordwrap, "MULTITEXT");

  if (IupGetInt(item_wordwrap, "VALUE"))
    IupSetAttribute(multitext, "WORDWRAP", "WORD");
  else
    IupSetAttribute(multitext, "WORDWRAP", "NONE");

  return IUP_DEFAULT;
```

```
}

int item_showwhite_action_cb(Ihandle* item_showwhite)
{
  Ihandle* multitext = IupGetDialogChild(item_showwhite, "MULTITEXT");

  if (IupGetInt(item_showwhite, "VALUE"))
    IupSetAttribute(multitext, "WHITESPACEVIEW", "VISIBLEALWAYS");
  else
    IupSetAttribute(multitext, "WHITESPACEVIEW", "INVISIBLE");

  return IUP_DEFAULT;
}

int item_toolbar_action_cb(Ihandle* item_toolbar)
{
  Ihandle* multitext = IupGetDialogChild(item_toolbar, "MULTITEXT");
  Ihandle* toolbar = IupGetChild(IupGetParent(multitext), 0);
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

  toggle_bar_visibility(item_toolbar, toolbar);

  IupConfigSetVariableStr(config, "MainWindow", "Toolbar", IupGetAttribute(item_too
  return IUP_DEFAULT;
}

int item_statusbar_action_cb(Ihandle* item_statusbar)
{
  Ihandle* multitext = IupGetDialogChild(item_statusbar, "MULTITEXT");
  Ihandle* statusbar = IupGetBrother(multitext);
  Ihandle* config = (Ihandle*)IupGetAttribute(multitext, "CONFIG");

  toggle_bar_visibility(item_statusbar, statusbar);

  IupConfigSetVariableStr(config, "MainWindow", "Statusbar", IupGetAttribute(item_s
  return IUP_DEFAULT;
}

int item_linenumber_action_cb(Ihandle* item_linenumber)
{
  Ihandle* multitext = IupGetDialogChild(item_linenumber, "MULTITEXT");
```

```
  if (IupGetInt(item_linenumber, "VALUE"))
  {
    IupSetInt(multitext, "MARGINWIDTH0", 0);
    IupSetAttribute(item_linenumber, "VALUE", "OFF");
  }
  else
  {
    IupSetInt(multitext, "MARGINWIDTH0", 50);
    IupSetAttribute(item_linenumber, "VALUE", "ON");
  }

  return IUP_DEFAULT;
}

int item_bookmark_action_cb(Ihandle* item_bookmark)
{
  Ihandle* multitext = IupGetDialogChild(item_bookmark, "MULTITEXT");

  if (IupGetInt(item_bookmark, "VALUE"))
  {
    IupSetInt(multitext, "MARGINWIDTH1", 0);
    IupSetAttribute(item_bookmark, "VALUE", "OFF");
  }
  else
  {
    IupSetInt(multitext, "MARGINWIDTH1", 20);
    IupSetAttribute(item_bookmark, "VALUE", "ON");
  }

  return IUP_DEFAULT;
}

int item_help_action_cb(void)
{
  IupHelp("http://www.tecgraf.puc-rio.br/iup");
  return IUP_DEFAULT;
}

int item_about_action_cb(void)
{
  IupMessage("About", "␣␣␣Scintilla␣Notepad\n\nAutors:\n␣␣␣Camilo␣Freire\n␣␣␣Antoni
  return IUP_DEFAULT;
```

```
}

/********************************* Main ****************************************/


Ihandle* create_main_dialog(Ihandle *config)
{
  Ihandle *dlg, *vbox, *multitext, *menu;
  Ihandle *sub_menu_file, *file_menu, *item_exit, *item_new, *item_open, *item_save
  Ihandle *sub_menu_edit, *edit_menu, *item_find, *item_find_next, *item_goto, *iten
  Ihandle *item_togglemark, *item_nextmark, *item_previousmark, *item_clearmarks, *
    *item_removeunmarked, *item_invertmarks, *item_tabtospace, *item_allspacetotab,
  Ihandle *item_trimleading, *item_trimtrailing, *item_trimtraillead, *item_eoltosp
  Ihandle *item_undo, *item_redo;
  Ihandle *case_menu, *item_uppercase, *item_lowercase;
  Ihandle *btn_cut, *btn_copy, *btn_paste, *btn_find, *btn_new, *btn_open, *btn_sav
  Ihandle *sub_menu_format, *format_menu, *item_font, *item_tab, *item_replace;
  Ihandle *sub_menu_help, *help_menu, *item_help, *item_about;
  Ihandle *sub_menu_view, *view_menu, *item_toolbar, *item_statusbar, *item_linenum
  Ihandle *zoom_menu, *item_zoomin, *item_zoomout, *item_restorezoom;
  Ihandle *lbl_statusbar, *toolbar_hb, *recent_menu;
  Ihandle *item_wordwrap, *item_showwhite;
  const char* font;

  multitext = IupScintilla();
  IupSetAttribute(multitext, "MULTILINE", "YES");
  IupSetAttribute(multitext, "EXPAND", "YES");
  IupSetAttribute(multitext, "NAME", "MULTITEXT");
  IupSetAttribute(multitext, "DIRTY", "NO");
  IupSetCallback(multitext, "CARET_CB", (Icallback)multitext_caret_cb);
  IupSetCallback(multitext, "VALUECHANGED_CB", (Icallback)multitext_valuechanged_cb
  IupSetCallback(multitext, "DROPFILES_CB", (Icallback)dropfiles_cb);
  IupSetCallback(multitext, "MARGINCLICK_CB", (Icallback)marginclick_cb);

  IupSetAttribute(multitext, "STYLEFGCOLOR34", "255␣0␣0");
  /* line numbers */
  IupSetInt(multitext, "MARGINWIDTH0", 30);
  IupSetAttribute(multitext, "MARGINSENSITIVE0", "YES");
  /* bookmarks */
  IupSetInt(multitext, "MARGINWIDTH1", 15);
  IupSetAttribute(multitext, "MARGINTYPE1", "SYMBOL");
```

```
  IupSetAttribute(multitext, "MARGINSENSITIVE1", "YES");
  IupSetAttribute(multitext, "MARGINMASKFOLDERS1", "NO");

  lbl_statusbar = IupLabel("Lin 1, Col 1");
  IupSetAttribute(lbl_statusbar, "NAME", "STATUSBAR");
  IupSetAttribute(lbl_statusbar, "EXPAND", "HORIZONTAL");
  IupSetAttribute(lbl_statusbar, "PADDING", "10x5");

  item_new = IupItem("&New\tCtrl+N", NULL);
  IupSetAttribute(item_new, "IMAGE", "IUP_FileNew");
  IupSetCallback(item_new, "ACTION", (Icallback)item_new_action_cb);
  btn_new = IupButton(NULL, NULL);
  IupSetAttribute(btn_new, "IMAGE", "IUP_FileNew");
  IupSetAttribute(btn_new, "FLAT", "Yes");
  IupSetCallback(btn_new, "ACTION", (Icallback)item_new_action_cb);
  IupSetAttribute(btn_new, "TIP", "New (Ctrl+N)");
  IupSetAttribute(btn_new, "CANFOCUS", "No");

  item_open = IupItem("&Open...\tCtrl+O", NULL);
  IupSetAttribute(item_open, "IMAGE", "IUP_FileOpen");
  IupSetCallback(item_open, "ACTION", (Icallback)item_open_action_cb);
  btn_open = IupButton(NULL, NULL);
  IupSetAttribute(btn_open, "IMAGE", "IUP_FileOpen");
  IupSetAttribute(btn_open, "FLAT", "Yes");
  IupSetCallback(btn_open, "ACTION", (Icallback)item_open_action_cb);
  IupSetAttribute(btn_open, "TIP", "Open (Ctrl+O)");
  IupSetAttribute(btn_open, "CANFOCUS", "No");

  item_save = IupItem("&Save\tCtrl+S", NULL);
  IupSetAttribute(item_save, "NAME", "ITEM_SAVE");
  IupSetAttribute(item_save, "IMAGE", "IUP_FileSave");
  IupSetCallback(item_save, "ACTION", (Icallback)item_save_action_cb);
  btn_save = IupButton(NULL, NULL);
  IupSetAttribute(btn_save, "IMAGE", "IUP_FileSave");
  IupSetAttribute(btn_save, "FLAT", "Yes");
  IupSetCallback(btn_save, "ACTION", (Icallback)item_save_action_cb);
  IupSetAttribute(btn_save, "TIP", "Save (Ctrl+S)");
  IupSetAttribute(btn_save, "CANFOCUS", "No");

  item_saveas = IupItem("Save &As...", NULL);
  IupSetAttribute(item_saveas, "NAME", "ITEM_SAVEAS");
  IupSetCallback(item_saveas, "ACTION", (Icallback)item_saveas_action_cb);
```

```
item_revert = IupItem("&Revert", NULL);
IupSetAttribute(item_revert, "NAME", "ITEM_REVERT");
IupSetCallback(item_revert, "ACTION", (Icallback)item_revert_action_cb);

item_exit = IupItem("E&xit", NULL);
IupSetCallback(item_exit, "ACTION", (Icallback)item_exit_action_cb);

item_find = IupItem("&Find...\tCtrl+F", NULL);
IupSetAttribute(item_find, "IMAGE", "IUP_EditFind");
IupSetCallback(item_find, "ACTION", (Icallback)item_find_action_cb);
btn_find = IupButton(NULL, NULL);
IupSetAttribute(btn_find, "IMAGE", "IUP_EditFind");
IupSetAttribute(btn_find, "FLAT", "Yes");
IupSetCallback(btn_find, "ACTION", (Icallback)item_find_action_cb);
IupSetAttribute(btn_find, "TIP", "Find (Ctrl+F)");
IupSetAttribute(btn_find, "CANFOCUS", "No");

item_find_next = IupItem("Find &Next\tF3", NULL);
IupSetAttribute(item_find_next, "NAME", "ITEM_FINDNEXT");
IupSetCallback(item_find_next, "ACTION", (Icallback)find_next_action_cb);

item_replace = IupItem("&Replace...\tCtrl+H", NULL);
IupSetCallback(item_replace, "ACTION", (Icallback)item_replace_action_cb);

item_cut = IupItem("Cu&t\tCtrl+X", NULL);
IupSetAttribute(item_cut, "NAME", "ITEM_CUT");
IupSetAttribute(item_cut, "IMAGE", "IUP_EditCut");
IupSetCallback(item_cut, "ACTION", (Icallback)item_cut_action_cb);
btn_cut = IupButton(NULL, NULL);
IupSetAttribute(btn_cut, "IMAGE", "IUP_EditCut");
IupSetAttribute(btn_cut, "FLAT", "Yes");
IupSetCallback(btn_cut, "ACTION", (Icallback)item_cut_action_cb);
IupSetAttribute(btn_cut, "TIP", "Cut (Ctrl+X)");
IupSetAttribute(btn_cut, "CANFOCUS", "No");

item_copy = IupItem("&Copy\tCtrl+C", NULL);
IupSetAttribute(item_copy, "NAME", "ITEM_COPY");
IupSetAttribute(item_copy, "IMAGE", "IUP_EditCopy");
IupSetCallback(item_copy, "ACTION", (Icallback)item_copy_action_cb);
btn_copy = IupButton(NULL, NULL);
IupSetAttribute(btn_copy, "IMAGE", "IUP_EditCopy");
```

```
  IupSetAttribute(btn_copy, "FLAT", "Yes");
  IupSetCallback(btn_copy, "ACTION", (Icallback)item_copy_action_cb);
  IupSetAttribute(btn_copy, "TIP", "Copy (Ctrl+C)");
  IupSetAttribute(btn_copy, "CANFOCUS", "No");

  item_paste = IupItem("&Paste\tCtrl+V", NULL);
  IupSetAttribute(item_paste, "NAME", "ITEM_PASTE");
  IupSetAttribute(item_paste, "IMAGE", "IUP_EditPaste");
  IupSetCallback(item_paste, "ACTION", (Icallback)item_paste_action_cb);
  btn_paste = IupButton(NULL, NULL);
  IupSetAttribute(btn_paste, "IMAGE", "IUP_EditPaste");
  IupSetAttribute(btn_paste, "FLAT", "Yes");
  IupSetCallback(btn_paste, "ACTION", (Icallback)item_paste_action_cb);
  IupSetAttribute(btn_paste, "TIP", "Paste (Ctrl+V)");
  IupSetAttribute(btn_paste, "CANFOCUS", "No");

  item_delete = IupItem("&Delete\tDel", NULL);
  IupSetAttribute(item_delete, "IMAGE", "IUP_EditErase");
  IupSetAttribute(item_delete, "NAME", "ITEM_DELETE");
  IupSetCallback(item_delete, "ACTION", (Icallback)item_delete_action_cb);

  item_select_all = IupItem("Select &All\tCtrl+A", NULL);
  IupSetCallback(item_select_all, "ACTION", (Icallback)item_select_all_action_cb);

  item_undo = IupItem("Undo\tCtrl+Z", NULL);
  IupSetCallback(item_undo, "ACTION", (Icallback)item_undo_action_cb);

  item_redo = IupItem("Redo\tCtrl+Y", NULL);
  IupSetCallback(item_redo, "ACTION", (Icallback)item_redo_action_cb);

  item_uppercase = IupItem("UPPERCASE\tCtrl+Shift+U", NULL);
  IupSetCallback(item_uppercase, "ACTION", (Icallback)item_uppercase_action_cb);

  item_lowercase = IupItem("lowercase\tCtrl+U", NULL);
  IupSetCallback(item_lowercase, "ACTION", (Icallback)item_lowercase_action_cb);

  item_goto = IupItem("&Go To...\tCtrl+G", NULL);
  IupSetCallback(item_goto, "ACTION", (Icallback)item_goto_action_cb);

  item_gotombrace = IupItem("Go To Matching Brace\tCtrl+B", NULL);
  IupSetCallback(item_gotombrace, "ACTION", (Icallback)item_gotombrace_action_cb);
```

```
item_togglemark = IupItem("Toggle Bookmark\tCtrl+F2", NULL);
IupSetCallback(item_togglemark, "ACTION", (Icallback)item_togglemark_action_cb);

item_nextmark = IupItem("Next Bookmark\tF2", NULL);
IupSetCallback(item_nextmark, "ACTION", (Icallback)item_nextmark_action_cb);

item_previousmark = IupItem("Previous Bookmark\tShift+F2", NULL);
IupSetCallback(item_previousmark, "ACTION", (Icallback)item_previousmark_action_c

item_clearmarks = IupItem("Clear All Bookmarks", NULL);
IupSetCallback(item_clearmarks, "ACTION", (Icallback)item_clearmarks_action_cb);

item_copymarked = IupItem("Copy Bookmarked Lines", NULL);
IupSetCallback(item_copymarked, "ACTION", (Icallback)item_copymarked_action_cb);

item_cutmarked = IupItem("Cut Bookmarked Lines", NULL);
IupSetCallback(item_cutmarked, "ACTION", (Icallback)item_cutmarked_action_cb);

item_pastetomarked = IupItem("Paste to (Replace) Bookmarked Lines", NULL);
IupSetCallback(item_pastetomarked, "ACTION", (Icallback)item_pastetomarked_action

item_removemarked = IupItem("Remove Bookmarked Lines", NULL);
IupSetCallback(item_removemarked, "ACTION", (Icallback)item_removemarked_action_c

item_removeunmarked = IupItem("Remove unmarked Lines", NULL);
IupSetCallback(item_removeunmarked, "ACTION", (Icallback)item_removeunmarked_acti

item_invertmarks = IupItem("Inverse Bookmark", NULL);
IupSetCallback(item_invertmarks, "ACTION", (Icallback)item_invertmarks_action_cb)

item_trimtrailing = IupItem("Trim Trailing Space", NULL);
IupSetCallback(item_trimtrailing, "ACTION", (Icallback)item_trimtrailing_action_c

item_trimtraillead = IupItem("Trim Trailing and Leading Space", NULL);
IupSetCallback(item_trimtraillead, "ACTION", (Icallback)item_trimtraillead_action

item_eoltospace = IupItem("EOL to Space", NULL);
IupSetCallback(item_eoltospace, "ACTION", (Icallback)item_eoltospace_action_cb);

item_removespaceeol = IupItem("Remove Unnecessary Blanks and EOL", NULL);
IupSetCallback(item_removespaceeol, "ACTION", (Icallback)item_removespaceeol_acti
```

```
item_trimleading = IupItem("Trim Leading Space", NULL);
IupSetCallback(item_trimleading, "ACTION", (Icallback)item_trimleading_action_cb)

item_tabtospace = IupItem("TAB to Space", NULL);
IupSetCallback(item_tabtospace, "ACTION", (Icallback)item_tabtospace_action_cb);

item_allspacetotab = IupItem("Space to TAB (All)", NULL);
IupSetCallback(item_allspacetotab, "ACTION", (Icallback)item_allspacetotab_action_

item_leadingspacetotab = IupItem("Space to TAB (Leading)", NULL);
IupSetCallback(item_leadingspacetotab, "ACTION", (Icallback)item_leadingspacetotab

item_zoomin = IupItem("Zoom In\tCtrl_Num +", NULL);
IupSetCallback(item_zoomin, "ACTION", (Icallback)item_zoomin_action_cb);

item_zoomout = IupItem("Zoom Out\tCtrl_Num -", NULL);
IupSetCallback(item_zoomout, "ACTION", (Icallback)item_zoomout_action_cb);

item_restorezoom = IupItem("Restore Default Zoom\tCtrl_Num /", NULL);
IupSetCallback(item_restorezoom, "ACTION", (Icallback)item_restorezoom_action_cb)

item_wordwrap = IupItem("Word Wrap", NULL);
IupSetCallback(item_wordwrap, "ACTION", (Icallback)item_wordwrap_action_cb);
IupSetAttribute(item_wordwrap, "AUTOTOGGLE", "YES");

item_showwhite = IupItem("Show White Spaces", NULL);
IupSetCallback(item_showwhite, "ACTION", (Icallback)item_showwhite_action_cb);
IupSetAttribute(item_showwhite, "AUTOTOGGLE", "YES");

item_toolbar = IupItem("&Toobar", NULL);
IupSetCallback(item_toolbar, "ACTION", (Icallback)item_toolbar_action_cb);
IupSetAttribute(item_toolbar, "VALUE", "ON");

item_statusbar = IupItem("&Statusbar", NULL);
IupSetCallback(item_statusbar, "ACTION", (Icallback)item_statusbar_action_cb);
IupSetAttribute(item_statusbar, "VALUE", "ON");

item_linenumber = IupItem("Display Line Numbers", NULL);
IupSetCallback(item_linenumber, "ACTION", (Icallback)item_linenumber_action_cb);
IupSetAttribute(item_linenumber, "VALUE", "ON");

item_bookmark = IupItem("Display Bookmarks", NULL);
```

```
  IupSetCallback(item_bookmark, "ACTION", (Icallback)item_bookmark_action_cb);
  IupSetAttribute(item_bookmark, "VALUE", "ON");

  item_font = IupItem("&Font...", NULL);
  IupSetCallback(item_font, "ACTION", (Icallback)item_font_action_cb);

  item_tab = IupItem("Tab...", NULL);
  IupSetCallback(item_tab, "ACTION", (Icallback)item_tab_action_cb);

  item_help = IupItem("&Help...", NULL);
  IupSetCallback(item_help, "ACTION", (Icallback)item_help_action_cb);

  item_about = IupItem("&About...", NULL);
  IupSetCallback(item_about, "ACTION", (Icallback)item_about_action_cb);

  recent_menu = IupMenu(NULL);

  file_menu = IupMenu(
    item_new,
    item_open,
    item_save,
    item_saveas,
    item_revert,
    IupSeparator(),
    IupSubmenu("Recent␣&Files", recent_menu),
    item_exit,
    NULL);
  edit_menu = IupMenu(
    item_undo,
    item_redo,
    IupSeparator(),
    item_cut,
    item_copy,
    item_paste,
    item_delete,
    IupSeparator(),
    item_find,
    item_find_next,
    item_replace,
    item_goto,
    item_gotombrace,
    IupSeparator(),
```

```
    IupSubmenu("Bookmarks", IupMenu(item_togglemark,
      item_nextmark,
      item_previousmark,
      item_clearmarks,
      item_cutmarked,
      item_copymarked,
      item_pastetomarked,
      item_removemarked,
      item_removeunmarked,
      item_invertmarks,
      NULL)),
    IupSubmenu("Blank Operations", IupMenu(
      item_trimtrailing,
      item_trimleading,
      item_trimtraillead,
      item_eoltospace,
      item_removespaceeol,
      IupSeparator(),
      item_tabtospace,
      item_allspacetotab,
      item_leadingspacetotab,
      NULL)),
    IupSubmenu("Convert Case to", case_menu = IupMenu(
      item_uppercase,
      item_lowercase,
      NULL)),
    IupSeparator(),
    item_select_all,
    NULL);
  format_menu = IupMenu(
    item_font,
    item_tab,
    NULL);
  view_menu = IupMenu(
    IupSubmenu("Zoom", zoom_menu = IupMenu(
    item_zoomin,
    item_zoomout,
    item_restorezoom,
    NULL)),
    item_wordwrap,
    item_showwhite,
    IupSeparator(),
```

```
    item_toolbar,
    item_statusbar,
    item_linenumber,
    item_bookmark,
    NULL);
  help_menu = IupMenu(
    item_help,
    item_about,
    NULL);

  IupSetCallback(file_menu, "OPEN_CB", (Icallback)file_menu_open_cb);
  IupSetCallback(edit_menu, "OPEN_CB", (Icallback)edit_menu_open_cb);

  sub_menu_file = IupSubmenu("&File", file_menu);
  sub_menu_edit = IupSubmenu("&Edit", edit_menu);
  sub_menu_format = IupSubmenu("F&ormat", format_menu);
  sub_menu_view = IupSubmenu("&View", view_menu);
  sub_menu_help = IupSubmenu("&Help", help_menu);

  menu = IupMenu(
    sub_menu_file,
    sub_menu_edit,
    sub_menu_format,
    sub_menu_view,
    sub_menu_help,
    NULL);

  toolbar_hb = IupHbox(
    btn_new,
    btn_open,
    btn_save,
    IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
    btn_cut,
    btn_copy,
    btn_paste,
    IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
    btn_find,
    NULL);
  IupSetAttribute(toolbar_hb, "MARGIN", "5x5");
  IupSetAttribute(toolbar_hb, "GAP", "2");

  vbox = IupVbox(
```

```
    toolbar_hb,
    multitext,
    lbl_statusbar,
    NULL);

  dlg = IupDialog(vbox);
  IupSetAttributeHandle(dlg, "MENU", menu);
  IupSetAttribute(dlg, "SIZE", "HALFxHALF");
  IupSetCallback(dlg, "CLOSE_CB", (Icallback)item_exit_action_cb);
  IupSetCallback(dlg, "DROPFILES_CB", (Icallback)dropfiles_cb);

  IupSetCallback(dlg, "K_cN", (Icallback)item_new_action_cb);
  IupSetCallback(dlg, "K_cO", (Icallback)item_open_action_cb);
  IupSetCallback(dlg, "K_cS", (Icallback)item_save_action_cb);
  IupSetCallback(dlg, "K_cF", (Icallback)item_find_action_cb);
  IupSetCallback(dlg, "K_cH", (Icallback)item_replace_action_cb);  /* replace syster
  IupSetCallback(dlg, "K_cG", (Icallback)item_goto_action_cb);
  IupSetCallback(dlg, "K_cB", (Icallback)item_gotombrace_action_cb);
  IupSetCallback(dlg, "K_cF2", (Icallback)item_togglemark_action_cb);
  IupSetCallback(dlg, "K_F2", (Icallback)item_nextmark_action_cb);
  IupSetCallback(dlg, "K_sF2", (Icallback)item_previousmark_action_cb);
  IupSetCallback(dlg, "K_F3", (Icallback)find_next_action_cb);
  IupSetCallback(dlg, "K_cF3", (Icallback)selection_find_next_action_cb);
  IupSetCallback(dlg, "K_cV", (Icallback)item_paste_action_cb);  /* replace system
  IupSetCallback(dlg, "K_c+", (Icallback)item_zoomin_action_cb);
  IupSetCallback(dlg, "K_c-", (Icallback)item_zoomout_action_cb);
  IupSetCallback(dlg, "K_c/", (Icallback)item_restorezoom_action_cb);
  IupSetCallback(dlg, "K_cU", (Icallback)item_case_action_cb);
  /* Ctrl+C, Ctrl+X, Ctrl+A, Del, already implemented inside IupText */

  /* parent for pre-defined dialogs in closed functions (IupMessage and IupAlarm) *
  IupSetAttributeHandle(NULL, "PARENTDIALOG", dlg);

  /* Initialize variables from the configuration file */

  IupConfigRecentInit(config, recent_menu, config_recent_cb, 10);

  font = IupConfigGetVariableStr(config, "MainWindow", "Font");
  if (font)
    IupSetStrAttribute(multitext, "FONT", font);

  IupSetAttribute(multitext, "WORDWRAPVISUALFLAGS", "MARGIN");
```

```
  /* line numbers */
  IupSetAttributeId(multitext, "MARKERFGCOLOR", 0, "0 0 255");
  IupSetAttributeId(multitext, "MARKERBGCOLOR", 0, "0 0 255");
  IupSetAttributeId(multitext, "MARKERALPHA", 0, "80");
  IupSetAttributeId(multitext, "MARKERSYMBOL", 0, "CIRCLE");
  /* bookmarks */
  IupSetIntId(multitext, "MARGINMASK", 1, 0x000005);
  IupSetAttributeId(multitext, "MARKERFGCOLOR", 1, "255 0 0");
  IupSetAttributeId(multitext, "MARKERBGCOLOR", 1, "255 0 0");
  IupSetAttributeId(multitext, "MARKERALPHA", 1, "80");
  IupSetAttributeId(multitext, "MARKERSYMBOL", 1, "CIRCLE");

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Toolbar", 1))
  {
    IupSetAttribute(item_toolbar, "VALUE", "OFF");

    IupSetAttribute(toolbar_hb, "FLOATING", "YES");
    IupSetAttribute(toolbar_hb, "VISIBLE", "NO");
  }

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Statusbar", 1))
  {
    IupSetAttribute(item_statusbar, "VALUE", "OFF");

    IupSetAttribute(lbl_statusbar, "FLOATING", "YES");
    IupSetAttribute(lbl_statusbar, "VISIBLE", "NO");
  }

  IupSetAttribute(dlg, "CONFIG", (char*)config);

  return dlg;
}

int main(int argc, char **argv)
{
  Ihandle *main_dialog;
  Ihandle *config;

  IupOpen(&argc, &argv);
  IupImageLibOpen();

  IupScintillaOpen();
```

```
  config = IupConfig();
  IupSetAttribute(config, "APP_NAME", "scintilla_notepad");
  IupConfigLoad(config);

  main_dialog = create_main_dialog(config);

  /* show the dialog at the last position, with the last size */
  IupConfigDialogShow(config, main_dialog, "MainWindow");

  /* initialize the current file */
  new_file(main_dialog);

  /* open a file from the command line (allow file association in Windows) */
  if (argc > 1 && argv[1])
  {
    const char* filename = argv[1];
    open_file(main_dialog, filename);
  }

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

在下一章，我们会介绍使用 *Tecgraf* 库在画布上绘制几何图元，也就是说，我们在下一章会构建一个画图程序。

4 简易画板

4.1 读取和保存图像

在前面的章节，我们看到了如何使用 **IUP** 来构建一个简单的记事本程序。在本章，我们将修改上一章节的代码来构建一个简易的画图程序。首先，我们需要定义一个结构来表示图像，还需要实现一些函数来读写常见的图像文件。在这里为了把关注点放在 **IUP**，我们使用一个叫做 **IM** 的库来进行这些工作。

IM 是一个数字图像处理库。它的主要目标是为科研程序提供一个处理图像的简单 API 和抽象。使用 **IM** 库，我们需要包含一些新的头文件：*iup.h* 是 **IM** 的主头文件，*im_image.h* 用于处理创建，载入，属性操作和图像存储。*im_convert.h* 用于处理不同文件类型之间的转换。*iup_im.h* 允许我们使用 **IUP** 来读取和存储图像。我们需要在编译时连接 **IM** 库。如果使用 Lua，我们只需要包含两个包：*imlua* 和 *iupluaim*。

我们还添加了一些函数，比如 *str_fileext* 函数用于提取文件名中的文件后缀名。*show_error* 函数会在屏幕上显示一条错误信息。*show_file_error* 函数调用 *show_error* 函数来告知用户在读取图

像文件时发生了什么错误。*set_file_format* 函数根据图像文件后缀名来设置图像格式。*select_file* 函数用于选择一个文件。相对之前的文本处理，*edit* 菜单的一些项目被移除。

read_file 和 *write_file* 现在使用 **IM** 函数。*imFileImageLoadBitmap* 和 *imFileImageSave* 函数用来读取和写入图像文件。它们使用一个叫做 *imImage* 的结构。需要注意，在读取图像文件时，除了错误处理，将图像转换为 RGB 形式是必要的。尽管 **IM** 可以处理多种文件类型，但是为了简化处理，我们把读取的文件都转换为 RGB 的形式。在写入文件时，我们则使用读取时使用的文件格式。图像文件的格式是通过 *imImageGetAttribString* 函数来获取的。

我们创建一个画布，然后使用它的 *IMAGE* 属性来保存图像，设置图像之后，需要记得销毁之前的图像。*set_file_format* 函数用来设置图像的格式。默认情况下，我们使用目前最流行的 JPEG 格式。

另一个变化是剪贴板的使用。我们使用 *NATIVEIMAGE* 属性进行图像的复制和粘贴。这个属性需要一个指定的格式。我们可以调用 *IupGetImageNativeHandle* 函数来从 *imImage* 产生这种格式。在读取图像文件和粘贴剪贴板后，我们需要调用 *IupUpdate* 函数重绘画布。*IupUpdate* 函数会调用重绘的回调函数。需要注意，剪贴板的内容来源可以是其它程序，所以剪贴板上的图像可能是任意的类型。由于我们的程序仅仅工作了 RGB 格式下，我们可能需要使用 *imImageRemoveAlpha* 函数来移除图像的 Alpha 通道，使用 *imConvertColorSpace* 函数来转换图像的颜色空间。

存储图像文件，除了需要文件名，我们还需要存储的格式。我们使用 *set_file_format* 函数来完成这个工作。默认情况下，我们使用目前最为流行的 JPEG 格式。

我们编写了一个 *select_file* 函数来用于选择打开或保存的文件地址。

在本例中，我们使用了 *IupGetParam* 预定义对话框。它被我们用来设置新图像的高度和宽度大小。程序在获取这些信息之后，调用 *imImageCreate* 函数来创建图像。

需要注意图像绘制并没有在第一个例子中实现，它是下一节我们要解释的内容。

在 Lua 形式下，我们需要包含 *imlua* 和 *iupluaim* 两个包。和 **IUP** 一样，**IM** 也是一个 **Lua** 包。我们在 Lua 下使用 *im.* 来访问 **IM** 包。Lua 下的 *im.FileImageLoadBitmap* 等价于 C 语言的 *imFileImageLoadBitmap*。C 语言形式下，这些函数有一个参数来接受图像信息，在 Lua 形式下，这些函数则是图像的方法，调用形式如 *image:RemoveAlpha*，*image:Destroy* 等。尽管 Lua 可以自动进行垃圾回收，但图像数据占用空间较大，依靠自动收集，内存消耗会成为一个问题，所以我们应该使用 *image:Destroy* 来释放不使用的图像。

C语言

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <iup.h>
#include <iup_config.h>
#include <im.h>
#include <im_image.h>
#include <im_convert.h>
#include <iupim.h>

/********************************** Utilities *****************************************
```

```
const char* str_filetitle(const char *filename)
{
  /* Start at the last character */
  int len = (int)strlen(filename);
  int offset = len - 1;
  while (offset != 0)
  {
    if (filename[offset] == '\\' || filename[offset] == '/')
    {
      offset++;
      break;
    }
    offset--;
  }
  return filename + offset;
}

const char* str_fileext(const char *filename)
{
  /* Start at the last character */
  int len = (int)strlen(filename);
  int offset = len - 1;
  while (offset != 0)
  {
    if (filename[offset] == '\\' || filename[offset] == '/')
      break;

    if (filename[offset] == '.')
    {
      offset++;
      return filename + offset;
    }
    offset--;
  }
  return NULL;
}

int str_compare(const char *l, const char *r, int casesensitive)
{
  if (!l || !r)
```

```
    return 0;

  while (*l && *r)
  {
    int diff;
    char l_char = *l,
      r_char = *r;

    /* compute the difference of both characters */
    if (casesensitive)
      diff = l_char - r_char;
    else
      diff = tolower((int)l_char) - tolower((int)r_char);

    /* if they differ we have a result */
    if (diff != 0)
      return 0;

    /* otherwise process the next characters */
    ++l;
    ++r;
  }

  /* check also for terminator */
  if (*l == *r)
    return 1;

  if (*r == 0)
    return 1;  /* if second string is at terminator, then it is partially equal */

  return 0;
}

void show_error(const char* message, int is_error)
{
  Ihandle* dlg = IupMessageDlg();
  IupSetStrAttribute(dlg, "PARENTDIALOG", IupGetGlobal("PARENTDIALOG"));
  IupSetAttribute(dlg, "DIALOGTYPE", is_error ? "ERROR" : "WARNING");
  IupSetAttribute(dlg, "BUTTONS", "OK");
  IupSetStrAttribute(dlg, "TITLE", is_error ? "Error" : "Warning");
  IupSetStrAttribute(dlg, "VALUE", message);
  IupPopup(dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
```

```
  IupDestroy(dlg);
}

void show_file_error(int error)
{
  switch (error)
  {
  case IM_ERR_OPEN:
    show_error("Error Opening File.", 1);
    break;
  case IM_ERR_MEM:
    show_error("Insufficient memory.", 1);
    break;
  case IM_ERR_ACCESS:
    show_error("Error Accessing File.", 1);
    break;
  case IM_ERR_DATA:
    show_error("Image type not Supported.", 1);
    break;
  case IM_ERR_FORMAT:
    show_error("Invalid Format.", 1);
    break;
  case IM_ERR_COMPRESS:
    show_error("Invalid or unsupported compression.", 1);
    break;
  default:
    show_error("Unknown Error.", 1);
  }
}

imImage* read_file(const char* filename)
{
  int error;
  imImage* image = imFileImageLoadBitmap(filename, 0, &error);
  if (error)
    show_file_error(error);
  else
  {
    /* we are going to support only RGB images with no alpha */
    imImageRemoveAlpha(image);
    if (image->color_space != IM_RGB)
    {
```

```
      imImage* new_image = imImageCreateBased(image, -1, -1, IM_RGB, -1);
      imConvertColorSpace(image, new_image);
      imImageDestroy(image);

      image = new_image;
    }
  }
  return image;
}

int write_file(const char* filename, const imImage* image)
{
  const char* format = imImageGetAttribString(image, "FileFormat");
  int error = imFileImageSave(filename, format, image);
  if (error)
  {
    show_file_error(error);
    return 0;
  }
  return 1;
}

void new_file(Ihandle* ih, imImage* image)
{
  Ihandle* dlg = IupGetDialog(ih);
  Ihandle* canvas = IupGetDialogChild(dlg, "CANVAS");
  imImage* old_image = (imImage*)IupGetAttribute(canvas, "IMAGE");

  IupSetAttribute(dlg, "TITLE", "Untitled - Simple Paint");
  IupSetAttribute(canvas, "FILENAME", NULL);
  IupSetAttribute(canvas, "DIRTY", "NO");

  IupSetAttribute(canvas, "IMAGE", (char*)image);

  IupUpdate(canvas);

  if (old_image)
    imImageDestroy(old_image);
}

void check_new_file(Ihandle* dlg)
{
```

```
  Ihandle* canvas = IupGetDialogChild(dlg, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  if (!image)
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
    int width = IupConfigGetVariableIntDef(config, "NewImage", "Width", 640);
    int height = IupConfigGetVariableIntDef(config, "NewImage", "Height", 480);

    image = imImageCreate(width, height, IM_RGB, IM_BYTE);

    new_file(dlg, image);
  }
}

void open_file(Ihandle* ih, const char* filename)
{
  imImage* image = read_file(filename);
  if (image)
  {
    Ihandle* dlg = IupGetDialog(ih);
    Ihandle* canvas = IupGetDialogChild(dlg, "CANVAS");
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
    imImage* old_image = (imImage*)IupGetAttribute(canvas, "IMAGE");

    IupSetfAttribute(dlg, "TITLE", "%s - Simple Paint", str_filetitle(filename));
    IupSetStrAttribute(canvas, "FILENAME", filename);
    IupSetAttribute(canvas, "DIRTY", "NO");
    IupSetAttribute(canvas, "IMAGE", (char*)image);

    IupUpdate(canvas);

    if (old_image)
      imImageDestroy(old_image);

    IupConfigRecentUpdate(config, filename);
  }
}

void save_file(Ihandle* canvas)
{
  char* filename = IupGetAttribute(canvas, "FILENAME");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
```

```
  if (write_file(filename, image))
    IupSetAttribute(canvas, "DIRTY", "NO");
}

void set_file_format(imImage* image, const char* filename)
{
  const char* ext = str_fileext(filename);
  const char* format = "JPEG";
  if (str_compare(ext, "jpg", 0) || str_compare(ext, "jpeg", 0))
    format = "JPEG";
  else if (str_compare(ext, "bmp", 0))
    format = "BMP";
  else if (str_compare(ext, "png", 0))
    format = "PNG";
  else if (str_compare(ext, "tga", 0))
    format = "TGA";
  else if (str_compare(ext, "tif", 0) || str_compare(ext, "tiff", 0))
    format = "TIFF";
  imImageSetAttribString(image, "FileFormat", format);
}

void saveas_file(Ihandle* canvas, const char* filename)
{
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");

  set_file_format(image, filename);

  if (write_file(filename, image))
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");

    IupSetfAttribute(IupGetDialog(canvas), "TITLE", "%s - Simple Paint", str_fileti
    IupSetStrAttribute(canvas, "FILENAME", filename);
    IupSetAttribute(canvas, "DIRTY", "NO");

    IupConfigRecentUpdate(config, filename);
  }
}

int save_check(Ihandle* ih)
{
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
```

```
  if (IupGetInt(canvas, "DIRTY"))
  {
    switch (IupAlarm("Warning", "File not saved! Save it now?", "Yes", "No", "Cance
    {
    case 1:  /* save the changes and continue */
      save_file(canvas);
      break;
    case 2:  /* ignore the changes and continue */
      break;
    case 3:  /* cancel */
      return 0;
    }
  }
  return 1;
}

void toggle_bar_visibility(Ihandle* item, Ihandle* ih)
{
  if (IupGetInt(item, "VALUE"))
  {
    IupSetAttribute(ih, "FLOATING", "YES");
    IupSetAttribute(ih, "VISIBLE", "NO");
    IupSetAttribute(item, "VALUE", "OFF");
  }
  else
  {
    IupSetAttribute(ih, "FLOATING", "NO");
    IupSetAttribute(ih, "VISIBLE", "YES");
    IupSetAttribute(item, "VALUE", "ON");
  }

  IupRefresh(ih);  /* refresh the dialog layout */
}


/********************************* Callbacks ************************************


int dropfiles_cb(Ihandle* ih, const char* filename)
{
  if (save_check(ih))
    open_file(ih, filename);
```

```
  return IUP_DEFAULT;
}

int file_menu_open_cb(Ihandle* ih)
{
  Ihandle* item_revert = IupGetDialogChild(ih, "ITEM_REVERT");
  Ihandle* item_save = IupGetDialogChild(ih, "ITEM_SAVE");
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  char* filename = IupGetAttribute(canvas, "FILENAME");
  int dirty = IupGetInt(canvas, "DIRTY");

  if (dirty)
    IupSetAttribute(item_save, "ACTIVE", "YES");
  else
    IupSetAttribute(item_save, "ACTIVE", "NO");

  if (dirty && filename)
    IupSetAttribute(item_revert, "ACTIVE", "YES");
  else
    IupSetAttribute(item_revert, "ACTIVE", "NO");
  return IUP_DEFAULT;
}

int edit_menu_open_cb(Ihandle* ih)
{
  Ihandle *clipboard = IupClipboard();

  Ihandle *item_paste = IupGetDialogChild(ih, "ITEM_PASTE");

  if (!IupGetInt(clipboard, "IMAGEAVAILABLE"))
    IupSetAttribute(item_paste, "ACTIVE", "NO");
  else
    IupSetAttribute(item_paste, "ACTIVE", "YES");

  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int config_recent_cb(Ihandle* ih)
{
  if (save_check(ih))
```

```
  {
    char* filename = IupGetAttribute(ih, "TITLE");
    open_file(ih, filename);
  }
  return IUP_DEFAULT;
}

int item_new_action_cb(Ihandle* item_new)
{
  if (save_check(item_new))
  {
    Ihandle* canvas = IupGetDialogChild(item_new, "CANVAS");
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
    int width = IupConfigGetVariableIntDef(config, "NewImage", "Width", 640);
    int height = IupConfigGetVariableIntDef(config, "NewImage", "Height", 480);

    if (IupGetParam("New␣Image", NULL, NULL, "Width:␣%i[1,]\nHeight:␣%i[1,]\n", &wi
    {
      imImage* image = imImageCreate(width, height, IM_RGB, IM_BYTE);

      IupConfigSetVariableInt(config, "NewImage", "Width", width);
      IupConfigSetVariableInt(config, "NewImage", "Height", height);

      new_file(item_new, image);
    }
  }

  return IUP_DEFAULT;
}

int select_file(Ihandle* parent_dlg, int is_open)
{
  Ihandle* config = (Ihandle*)IupGetAttribute(parent_dlg, "CONFIG");
  Ihandle* canvas = IupGetDialogChild(parent_dlg, "CANVAS");
  const char* dir = IupConfigGetVariableStr(config, "MainWindow", "LastDirectory");

  Ihandle* filedlg = IupFileDlg();
  if (is_open)
    IupSetAttribute(filedlg, "DIALOGTYPE", "OPEN");
  else
  {
    IupSetAttribute(filedlg, "DIALOGTYPE", "SAVE");
```

```
    IupSetStrAttribute(filedlg, "FILE", IupGetAttribute(canvas, "FILENAME"));
  }
  IupSetAttribute(filedlg, "EXTFILTER", "Image␣Files|*.bmp;*.jpg;*.png;*.tif;*.tga|
  IupSetStrAttribute(filedlg, "DIRECTORY", dir);
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", parent_dlg);

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    if (is_open)
      open_file(parent_dlg, filename);
    else
      saveas_file(canvas, filename);

    dir = IupGetAttribute(filedlg, "DIRECTORY");
    IupConfigSetVariableStr(config, "MainWindow", "LastDirectory", dir);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_open_action_cb(Ihandle* item_open)
{
  if (!save_check(item_open))
    return IUP_DEFAULT;

  return select_file(IupGetDialog(item_open), 1);
}

int item_saveas_action_cb(Ihandle* item_saveas)
{
  return select_file(IupGetDialog(item_saveas), 0);
}

int item_save_action_cb(Ihandle* item_save)
{
  Ihandle* canvas = IupGetDialogChild(item_save, "CANVAS");
  char* filename = IupGetAttribute(canvas, "FILENAME");
  if (!filename)
    item_saveas_action_cb(item_save);
```

```
  else
  {
    /* test again because in can be called using the hot key */
    int dirty = IupGetInt(canvas, "DIRTY");
    if (dirty)
      save_file(canvas);
  }
  return IUP_DEFAULT;
}

int item_revert_action_cb(Ihandle* item_revert)
{
  Ihandle* canvas = IupGetDialogChild(item_revert, "CANVAS");
  char* filename = IupGetAttribute(canvas, "FILENAME");
  open_file(item_revert, filename);
  return IUP_DEFAULT;
}

int item_exit_action_cb(Ihandle* item_exit)
{
  Ihandle* dlg = IupGetDialog(item_exit);
  Ihandle* config = (Ihandle*)IupGetAttribute(dlg, "CONFIG");
  Ihandle* canvas = IupGetDialogChild(dlg, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");

  if (!save_check(item_exit))
    return IUP_IGNORE;  /* to abort the CLOSE_CB callback */

  if (image)
    imImageDestroy(image);

  IupConfigDialogClosed(config, dlg, "MainWindow");
  IupConfigSave(config);
  IupDestroy(config);
  return IUP_CLOSE;
}

int item_copy_action_cb(Ihandle* item_copy)
{
  Ihandle* canvas = IupGetDialogChild(item_copy, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  Ihandle *clipboard = IupClipboard();
```

```
  IupSetAttribute(clipboard, "NATIVEIMAGE", (char*)IupGetImageNativeHandle(image));
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_paste_action_cb(Ihandle* item_paste)
{
  if (save_check(item_paste))
  {
    Ihandle* canvas = IupGetDialogChild(item_paste, "CANVAS");
    imImage* old_image = (imImage*)IupGetAttribute(canvas, "IMAGE");

    Ihandle *clipboard = IupClipboard();
    imImage* image = IupGetNativeHandleImage(IupGetAttribute(clipboard, "NATIVEIMAG
    IupDestroy(clipboard);

    if (!image)
    {
      show_error("Invalid Clipboard Data", 1);
      return IUP_DEFAULT;
    }

    /* we are going to support only RGB images with no alpha */
    imImageRemoveAlpha(image);
    if (image->color_space != IM_RGB)
    {
      imImage* new_image = imImageCreateBased(image, -1, -1, IM_RGB, -1);
      imConvertColorSpace(image, new_image);
      imImageDestroy(image);

      image = new_image;
    }

    imImageSetAttribString(image, "FileFormat", "JPEG");

    IupSetAttribute(canvas, "DIRTY", "Yes");
    IupSetAttribute(canvas, "IMAGE", (char*)image);
    IupSetAttribute(canvas, "FILENAME", NULL);
    IupSetAttribute(IupGetDialog(canvas), "TITLE", "Untitled - Simple Paint");

    IupUpdate(canvas);
```

```
    if (old_image)
      imImageDestroy(old_image);
  }
  return IUP_DEFAULT;
}

int item_toolbar_action_cb(Ihandle* item_toolbar)
{
  Ihandle* canvas = IupGetDialogChild(item_toolbar, "CANVAS");
  Ihandle* toolbar = IupGetChild(IupGetParent(canvas), 0);
  Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");

  toggle_bar_visibility(item_toolbar, toolbar);

  IupConfigSetVariableStr(config, "MainWindow", "Toolbar", IupGetAttribute(item_too
  return IUP_DEFAULT;
}

int item_statusbar_action_cb(Ihandle* item_statusbar)
{
  Ihandle* canvas = IupGetDialogChild(item_statusbar, "CANVAS");
  Ihandle* statusbar = IupGetBrother(canvas);
  Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");

  toggle_bar_visibility(item_statusbar, statusbar);

  IupConfigSetVariableStr(config, "MainWindow", "Statusbar", IupGetAttribute(item_s
  return IUP_DEFAULT;
}

int item_help_action_cb(void)
{
  IupHelp("http://www.tecgraf.puc-rio.br/iup");
  return IUP_DEFAULT;
}

int item_about_action_cb(void)
{
  IupMessage("About", "␣␣␣Simple␣Paint\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣Scu
  return IUP_DEFAULT;
}
```

```
/********************************* Main ****************************************/


Ihandle* create_main_dialog(Ihandle *config)
{
  Ihandle *dlg, *vbox, *canvas, *menu;
  Ihandle *sub_menu_file, *file_menu, *item_exit, *item_new, *item_open, *item_save
  Ihandle *sub_menu_edit, *edit_menu, *item_copy, *item_paste;
  Ihandle *btn_copy, *btn_paste, *btn_new, *btn_open, *btn_save;
  Ihandle *sub_menu_help, *help_menu, *item_help, *item_about;
  Ihandle *sub_menu_view, *view_menu, *item_toolbar, *item_statusbar;
  Ihandle *statusbar, *toolbar, *recent_menu;

  canvas = IupCanvas(NULL);
  IupSetAttribute(canvas, "NAME", "CANVAS");
  IupSetAttribute(canvas, "DIRTY", "NO");
/* TODO: IupSetCallback(canvas, "ACTION", (Icallback)canvas_action_cb); */
  IupSetCallback(canvas, "DROPFILES_CB", (Icallback)dropfiles_cb);

  statusbar = IupLabel("(0, 0) = [0 0 0]");
  IupSetAttribute(statusbar, "NAME", "STATUSBAR");
  IupSetAttribute(statusbar, "EXPAND", "HORIZONTAL");
  IupSetAttribute(statusbar, "PADDING", "10x5");

  item_new = IupItem("&New\tCtrl+N", NULL);
  IupSetAttribute(item_new, "IMAGE", "IUP_FileNew");
  IupSetCallback(item_new, "ACTION", (Icallback)item_new_action_cb);
  btn_new = IupButton(NULL, NULL);
  IupSetAttribute(btn_new, "IMAGE", "IUP_FileNew");
  IupSetAttribute(btn_new, "FLAT", "Yes");
  IupSetCallback(btn_new, "ACTION", (Icallback)item_new_action_cb);
  IupSetAttribute(btn_new, "TIP", "New (Ctrl+N)");
  IupSetAttribute(btn_new, "CANFOCUS", "No");

  item_open = IupItem("&Open...\tCtrl+O", NULL);
  IupSetAttribute(item_open, "IMAGE", "IUP_FileOpen");
  IupSetCallback(item_open, "ACTION", (Icallback)item_open_action_cb);
  btn_open = IupButton(NULL, NULL);
  IupSetAttribute(btn_open, "IMAGE", "IUP_FileOpen");
  IupSetAttribute(btn_open, "FLAT", "Yes");
  IupSetCallback(btn_open, "ACTION", (Icallback)item_open_action_cb);
```

```
IupSetAttribute(btn_open, "TIP", "Open (Ctrl+O)");
IupSetAttribute(btn_open, "CANFOCUS", "No");

item_save = IupItem("&Save\tCtrl+S", NULL);
IupSetAttribute(item_save, "NAME", "ITEM_SAVE");
IupSetAttribute(item_save, "IMAGE", "IUP_FileSave");
IupSetCallback(item_save, "ACTION", (Icallback)item_save_action_cb);
btn_save = IupButton(NULL, NULL);
IupSetAttribute(btn_save, "IMAGE", "IUP_FileSave");
IupSetAttribute(btn_save, "FLAT", "Yes");
IupSetCallback(btn_save, "ACTION", (Icallback)item_save_action_cb);
IupSetAttribute(btn_save, "TIP", "Save (Ctrl+S)");
IupSetAttribute(btn_save, "CANFOCUS", "No");

item_saveas = IupItem("Save &As...", NULL);
IupSetAttribute(item_saveas, "NAME", "ITEM_SAVEAS");
IupSetCallback(item_saveas, "ACTION", (Icallback)item_saveas_action_cb);

item_revert = IupItem("&Revert", NULL);
IupSetAttribute(item_revert, "NAME", "ITEM_REVERT");
IupSetCallback(item_revert, "ACTION", (Icallback)item_revert_action_cb);

item_exit = IupItem("E&xit", NULL);
IupSetCallback(item_exit, "ACTION", (Icallback)item_exit_action_cb);

item_copy = IupItem("&Copy\tCtrl+C", NULL);
IupSetAttribute(item_copy, "NAME", "ITEM_COPY");
IupSetAttribute(item_copy, "IMAGE", "IUP_EditCopy");
IupSetCallback(item_copy, "ACTION", (Icallback)item_copy_action_cb);
btn_copy = IupButton(NULL, NULL);
IupSetAttribute(btn_copy, "IMAGE", "IUP_EditCopy");
IupSetAttribute(btn_copy, "FLAT", "Yes");
IupSetCallback(btn_copy, "ACTION", (Icallback)item_copy_action_cb);
IupSetAttribute(btn_copy, "TIP", "Copy (Ctrl+C)");
IupSetAttribute(btn_copy, "CANFOCUS", "No");

item_paste = IupItem("&Paste\tCtrl+V", NULL);
IupSetAttribute(item_paste, "NAME", "ITEM_PASTE");
IupSetAttribute(item_paste, "IMAGE", "IUP_EditPaste");
IupSetCallback(item_paste, "ACTION", (Icallback)item_paste_action_cb);
btn_paste = IupButton(NULL, NULL);
IupSetAttribute(btn_paste, "IMAGE", "IUP_EditPaste");
```

```
  IupSetAttribute(btn_paste, "FLAT", "Yes");
  IupSetCallback(btn_paste, "ACTION", (Icallback)item_paste_action_cb);
  IupSetAttribute(btn_paste, "TIP", "Paste (Ctrl+V)");
  IupSetAttribute(btn_paste, "CANFOCUS", "No");

  item_toolbar = IupItem("&Toobar", NULL);
  IupSetCallback(item_toolbar, "ACTION", (Icallback)item_toolbar_action_cb);
  IupSetAttribute(item_toolbar, "VALUE", "ON");

  item_statusbar = IupItem("&Statusbar", NULL);
  IupSetCallback(item_statusbar, "ACTION", (Icallback)item_statusbar_action_cb);
  IupSetAttribute(item_statusbar, "VALUE", "ON");

  item_help = IupItem("&Help...", NULL);
  IupSetCallback(item_help, "ACTION", (Icallback)item_help_action_cb);

  item_about = IupItem("&About...", NULL);
  IupSetCallback(item_about, "ACTION", (Icallback)item_about_action_cb);

  recent_menu = IupMenu(NULL);

  file_menu = IupMenu(
    item_new,
    item_open,
    item_save,
    item_saveas,
    item_revert,
    IupSeparator(),
    IupSubmenu("Recent &Files", recent_menu),
    item_exit,
    NULL);
  edit_menu = IupMenu(
    item_copy,
    item_paste,
    NULL);
  view_menu = IupMenu(
    item_toolbar,
    item_statusbar,
    NULL);
  help_menu = IupMenu(
    item_help,
    item_about,
```

```
    NULL);

  IupSetCallback(file_menu, "OPEN_CB", (Icallback)file_menu_open_cb);
  IupSetCallback(edit_menu, "OPEN_CB", (Icallback)edit_menu_open_cb);

  sub_menu_file = IupSubmenu("&File", file_menu);
  sub_menu_edit = IupSubmenu("&Edit", edit_menu);
  sub_menu_view = IupSubmenu("&View", view_menu);
  sub_menu_help = IupSubmenu("&Help", help_menu);

  menu = IupMenu(
    sub_menu_file,
    sub_menu_edit,
    sub_menu_view,
    sub_menu_help,
    NULL);

  toolbar = IupHbox(
    btn_new,
    btn_open,
    btn_save,
    IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
    btn_copy,
    btn_paste,
    NULL);
  IupSetAttribute(toolbar, "MARGIN", "5x5");
  IupSetAttribute(toolbar, "GAP", "2");

  vbox = IupVbox(
    toolbar,
    canvas,
    statusbar,
    NULL);

  dlg = IupDialog(vbox);
  IupSetAttributeHandle(dlg, "MENU", menu);
  IupSetAttribute(dlg, "SIZE", "HALFxHALF");
  IupSetCallback(dlg, "CLOSE_CB", (Icallback)item_exit_action_cb);
  IupSetCallback(dlg, "DROPFILES_CB", (Icallback)dropfiles_cb);

  IupSetCallback(dlg, "K_cN", (Icallback)item_new_action_cb);
  IupSetCallback(dlg, "K_cO", (Icallback)item_open_action_cb);
```

```
  IupSetCallback(dlg, "K_cS", (Icallback)item_save_action_cb);
  IupSetCallback(dlg, "K_cV", (Icallback)item_paste_action_cb);
  IupSetCallback(dlg, "K_cC", (Icallback)item_copy_action_cb);

  /* parent for pre-defined dialogs in closed functions (IupMessage and IupAlarm) *
  IupSetAttributeHandle(NULL, "PARENTDIALOG", dlg);

  /* Initialize variables from the configuration file */

  IupConfigRecentInit(config, recent_menu, config_recent_cb, 10);

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Toolbar", 1))
  {
    IupSetAttribute(item_toolbar, "VALUE", "OFF");

    IupSetAttribute(toolbar, "FLOATING", "YES");
    IupSetAttribute(toolbar, "VISIBLE", "NO");
  }

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Statusbar", 1))
  {
    IupSetAttribute(item_statusbar, "VALUE", "OFF");

    IupSetAttribute(statusbar, "FLOATING", "YES");
    IupSetAttribute(statusbar, "VISIBLE", "NO");
  }

  IupSetAttribute(dlg, "CONFIG", (char*)config);

  return dlg;
}

int main(int argc, char **argv)
{
  Ihandle *dlg;
  Ihandle *config;

  IupOpen(&argc, &argv);
  IupImageLibOpen();

  config = IupConfig();
  IupSetAttribute(config, "APP_NAME", "simple_paint");
```

```
  IupConfigLoad(config);

  dlg = create_main_dialog(config);

  /* show the dialog at the last position, with the last size */
  IupConfigDialogShow(config, dlg, "MainWindow");

  /* open a file from the command line (allow file association in Windows) */
  if (argc > 1 && argv[1])
  {
    const char* filename = argv[1];
    open_file(dlg, filename);
  }

  /* initialize the current file, if not already loaded */
  check_new_file(dlg);

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")
require("iupluaimglib")
require("imlua")
require("iupluaim")


--********************************* Utilities *********************************

function str_splitfilename(filename)
  return string.match(filename, "(.-)([^\\/]-%.?([^%.\\/]*))$")
end

function str_fileext(filename)
  local path, title, ext = str_splitfilename(filename)
  return ext
end

function str_filetitle(filename)
```

```
  local path, title, ext = str_splitfilename(filename)
  return title
end

function show_error(message, is_error)
  local dlg = iup.messagedlg{
    parentdialog = iup.GetGlobal("PARENTDIALOG"),
    buttons = "OK",
    value = message,
  }
  if (is_error) then
    dlg.dialogtype = "ERROR"
    dlg.title = "Error"
  else
    dlg.dialogtype = "WARNING"
    dlg.title = "Warning"
  end
  dlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)
  dlg:destroy()
end

function read_file(filename)
  local image, err = im.FileImageLoadBitmap(filename, 0)
  if (err) then
    show_error(im.ErrorStr(err), true)
  else
    -- we are going to support only RGB images with no alpha
    image:RemoveAlpha()
    if (image:ColorSpace() ~= im.RGB) then
      local new_image = im.ImageCreateBased(image, nil, nil, im.RGB, nil)

      im.ConvertColorSpace(image, new_image)
      image:Destroy()

      image = new_image
    end
  end
  return image
end

function write_file(filename, image)
  local format = image:GetAttribString("FileFormat")
```

```
  local err = im.FileImageSave(filename, format, image)
  if (err and err ~= im.ERR_NONE) then
    show_error(im.ErrorStr(err), true)
    return false
  end
  return true
end

function new_file(ih, image)
  local dlg = iup.GetDialog(ih)
  local canvas = dlg.canvas
  local old_image = canvas.image

  dlg.title = "Untitled - Simple Paint"
  canvas.filename = nil
  canvas.dirty = nil
  canvas.image = image

  iup.Update(canvas)

  if (old_image) then
    old_image:Destroy()
  end
end

function check_new_file(dlg)
  local canvas = dlg.canvas
  local image = canvas.image
  if (not image) then
    local config = canvas.config
    local width = config:GetVariableDef("NewImage", "Width", 640)
    local height = config:GetVariableDef("NewImage", "Height", 480)

    local image = im.ImageCreate(width, height, im.RGB, im.BYTE)

    new_file(dlg, image)
  end
end

function open_file(ih, filename)
  local image = read_file(filename)
  if (image) then
```

```
    local dlg = iup.GetDialog(ih)
    local canvas = dlg.canvas
    local config = canvas.config
    local old_image = canvas.image

    dlg.title = str_filetitle(filename).." - Simple Paint"
    canvas.filename = filename
    canvas.dirty = nil
    canvas.image = image

    iup.Update(canvas)

    if (old_image) then
      old_image:Destroy()
    end

    config:RecentUpdate(filename)
  end
end

function save_file(canvas)
  if (write_file(canvas.filename, canvas.image)) then
    canvas.dirty = nil
  end
end

function set_file_format(image, filename)
  local ext = str_fileext(filename)
  ext:lower()
  local format = "JPEG"
  if (ext == "jpg" or ext == "jpeg") then
    format = "JPEG"
  elseif (ext == "bmp") then
    format = "BMP"
  elseif (ext == "png") then
    format = "PNG"
  elseif (ext == "tga") then
    format = "TGA"
  elseif (ext == "tif" or ext == "tiff") then
    format = "TIFF"
  end
  image:SetAttribString("FileFormat", format)
```

```
end

function saveas_file(canvas, filename)
  local image = canvas.image

  set_file_format(image, filename)

  if (write_file(filename, image)) then
    local dlg = iup.GetDialog(canvas)
    local config = canvas.config

    dlg.title = str_filetitle(filename)..” – Simple Paint”
    canvas.filename = filename
    canvas.dirty = nil

    config:RecentUpdate(filename)
  end
end

function save_check(ih)
  local dlg = iup.GetDialog(ih)
  local canvas = dlg.canvas

  if (canvas.dirty) then
    local resp = iup.Alarm(”Warning”, ”File not saved! Save it now?”, ”Yes”, ”No”,
    if resp == 1 then -- save the changes and continue
      save_file(canvas)
    elseif resp == 3 then  -- cancel
      return false
    else  -- ignore the changes and continue
    end
  end
  return true
end

function toggle_bar_visibility(item, bar)
  if (item.value == ”ON”) then
    bar.floating = ”YES”
    bar.visible = ”NO”
    item.value = ”OFF”
  else
    bar.floating = ”NO”
```

```
    bar.visible = "YES"
    item.value = "ON"
  end
  iup.Refresh(bar)  -- refresh the dialog layout
end


--********************************* Main (Part 1/2) ****************************

-- create all the elements that will have callbacks in Lua prior to callbacks defin

config = iup.config{}
config.app_name = "simple_paint"
config:Load()

statusbar = iup.label{title = "(0, 0) = [0 0 0 0]", expand = "HORIZONTAL", paddin

canvas = iup.canvas{
  config = config,
  dirty = nil,
}

item_new = iup.item{title = "&New...\tCtrl+N", image = "IUP_FileNew"}
item_open = iup.item{title = "&Open...\tCtrl+O", image = "IUP_FileOpen"}
item_save = iup.item{title="&Save\tCtrl+S"}
item_saveas = iup.item{title="Save &As...", image = "IUP_FileSave"}
item_revert = iup.item{title="&Revert"}
item_exit = iup.item{title="E&xit"}
item_copy = iup.item{title="&Copy\tCtrl+C", image = "IUP_EditCopy"}
item_paste = iup.item{title="&Paste\tCtrl+V", image = "IUP_EditPaste"}
item_toolbar = iup.item{title="&Toobar", value="ON"}
item_statusbar = iup.item{title="&Statusbar", value="ON"}
item_help = iup.item{title="&Help..."}
item_about = iup.item{title="&About..."}

recent_menu = iup.menu{}

file_menu = iup.menu{
  item_new,
  item_open,
  item_save,
  item_saveas,
```

```
  item_revert,
  iup.separator{},
  iup.submenu{title="Recent &Files", recent_menu},
  item_exit
  }

edit_menu = iup.menu{
  item_copy,
  item_paste,
  }

view_menu = iup.menu{item_toolbar, item_statusbar}
help_menu = iup.menu{item_help, item_about}

sub_menu_file = iup.submenu{file_menu, title = "&File"}
sub_menu_edit = iup.submenu{edit_menu, title = "&Edit"}
sub_menu_view = iup.submenu{title = "&View", view_menu}
sub_menu_help = iup.submenu{help_menu, title = "&Help"}

menu = iup.menu{
  sub_menu_file,
  sub_menu_edit,
  sub_menu_view,
  sub_menu_help,
  }


--******************************** Callbacks ********************************


function canvas:dropfiles_cb(filename)
  if (save_check(self)) then
    open_file(self, filename)
  end
end

function file_menu:open_cb()
  if (canvas.dirty) then
    item_save.active = "YES"
  else
    item_save.active = "NO"
  end
```

```
  if (canvas.dirty and canvas.filename) then
    item_revert.active = "YES"
  else
    item_revert.active = "NO"
  end
end

function edit_menu:open_cb()
  local clipboard = iup.clipboard{}
  if (clipboard.imageavailable == "NO") then
    item_paste.active = "NO"
  else
    item_paste.active = "YES"
  end
  clipboard:destroy()
end

function config:recent_cb()
  if (save_check(self)) then
    local filename = self.title
    open_file(self, filename)
  end
end

function item_new:action()
  if save_check(self) then
    local width = config:GetVariableDef("NewImage", "Width", 640)
    local height = config:GetVariableDef("NewImage", "Height", 480)

    local ret, new_width, new_height = iup.GetParam("New Image", nil, "Width: %i[1,
    if (ret) then
      local new_image = im.ImageCreate(new_width, new_height, im.RGB, im.BYTE)

      config:SetVariable("NewImage", "Width", new_width)
      config:SetVariable("NewImage", "Height", new_height)

      new_file(item_new, new_image)
    end
  end
end

function select_file(parent_dlg, is_open)
```

```
  local filedlg = iup.filedlg{
    extfilter="Image Files|*.bmp;*.jpg;*.png;*.tif;*.tga|All Files|*.*|",
    parentdialog = parent_dlg,
    directory = config:GetVariable("MainWindow", "LastDirectory"),
    }

  if (is_open) then
    filedlg.dialogtype = "OPEN"
  else
    filedlg.dialogtype = "SAVE"
    filedlg.file = canvas.filename
  end

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    if (is_open) then
      open_file(parent_dlg, filename)
    else
      saveas_file(canvas, filename)
    end

    config:SetVariable("MainWindow", "LastDirectory", filedlg.directory)
  end

  filedlg:destroy()
end

function item_open:action()
  if not save_check(self) then
    return
  end

  select_file(dlg, true)
end

function item_saveas:action()
  select_file(dlg, false)
end

function item_save:action()
```

```
  if (not canvas.filename) then
    item_saveas:action()
  else
    -- test again because in can be called using the hot key
    if (canvas.dirty) then
      save_file(canvas)
    end
  end
end

function item_revert:action()
  open_file(self, canvas.filename)
end

function item_exit:action()
  local image = canvas.image

  if not save_check(self) then
    return iup.IGNORE  -- to abort the CLOSE_CB callback
  end

  if (image) then
    image:Destroy()
  end

  config:DialogClosed(iup.GetDialog(self), "MainWindow")
  config:Save()
  config:destroy()
  return iup.CLOSE
end

function item_copy:action()
  local clipboard = iup.clipboard{}
  -- must use iup.SetAttribute because it is an userdata
  iup.SetAttribute(clipboard, "NATIVEIMAGE", iup.GetImageNativeHandle(canvas.image))
  clipboard:destroy()
end

function item_paste:action()
  if save_check(self) then
    local clipboard = iup.clipboard{}
    local old_image = canvas.image
```

```
    local image = iup.GetNativeHandleImage(clipboard.nativeimage)

    -- we are going to support only RGB images with no alpha
    image:RemoveAlpha()
    if (image:ColorSpace() ~= im.RGB) then
      local new_image = im.ImageCreateBased(image, nil, nil, im.RGB, nil)

      im.ConvertColorSpace(image, new_image)
      image:Destroy()

      image = new_image
    end

    image:SetAttribString("FileFormat", "JPEG")

    canvas.dirty = "Yes"
    canvas.image = image
    canvas.filename = nil
    dlg.title = "Untitled - Simple Paint"

    iup.Update(canvas)

    if (old_image) then
      old_image:Destroy()
    end

    clipboard:destroy()
  end
end

function item_toolbar:action()
  toggle_bar_visibility(self, toolbar)
  config:SetVariable("MainWindow", "Toolbar", item_toolbar.value)
end

function item_statusbar:action()
  toggle_bar_visibility(self, statusbar)
  config:SetVariable("MainWindow", "Statusbar", item_statusbar.value)
end

function item_help:action()
```

```
  iup.Help("http://www.tecgraf.puc-rio.br/iup")
end

function item_about:action()
  iup.Message("About", "␣␣␣Simple␣Paint\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣Sc
end


--********************************* Main (Part 2/2) ****************************


btn_new = iup.button{image = "IUP_FileNew", flat = "Yes", action = item_new.action,
btn_open = iup.button{image = "IUP_FileOpen", flat = "Yes", action = item_open.actio
btn_save = iup.button{image = "IUP_FileSave", flat = "Yes", action = item_save.actio
btn_copy = iup.button{image =  "IUP_EditCopy", flat = "Yes", action = item_copy.act
btn_paste = iup.button{image = "IUP_EditPaste", flat = "Yes", action = item_paste.a

toolbar = iup.hbox{
  btn_new,
  btn_open,
  btn_save,
  iup.label{separator="VERTICAL"},
  btn_copy,
  btn_paste,
  margin = "5x5",
  gap = 2,
}

vbox = iup.vbox{
  toolbar,
  canvas,
  statusbar,
}

dlg = iup.dialog{
  vbox,
  title = "Simple␣Paint",
  size = "HALFxHALF",
  menu = menu,
  close_cb = item_exit.action,
  canvas = canvas,
  dropfiles_cb = canvas.dropfiles_cb,
```

```
}

function dlg:k_any(c)
  if (c == iup.K_cN) then
    item_new:action()
  elseif (c == iup.K_cO) then
    item_open:action()
  elseif (c == iup.K_cS) then
    item_save:action()
  elseif (c == iup.K_cV) then
    item_paste:action()
  elseif (c == iup.K_cC) then
    item_copy:action()
  end
end

-- parent for pre-defined dialogs in closed functions (IupMessage and IupAlarm)
iup.SetGlobal("PARENTDIALOG", iup.SetHandleName(dlg))

-- Initialize variables from the configuration file

config:RecentInit(recent_menu, 10)

show_statusbar = config:GetVariableDef("MainWindow", "Statusbar", "ON")
if (show_statusbar == "OFF") then
  item_statusbar.value = "OFF"
  statusbar.floating = "YES"
  statusbar.visible = "NO"
end

show_toolbar = config:GetVariableDef("MainWindow", "Toolbar", "ON")
if (show_toolbar == "OFF") then
  item_toolbar.value = "OFF"
  toolbar.floating = "YES"
  toolbar.visible = "NO"
end

-- show the dialog at the last position, with the last size
config:DialogShow(dlg, "MainWindow")

-- open a file from the command line (allow file association in Windows)
if (arg and arg[1]) then
```

```
  filename = arg[1]
  open_file(dlg, filename)
end

-- initialize the current file, if not already loaded
check_new_file(dlg)

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

4.2 使用 OpenGL 绘图

我们知道 **IUP** 是用来创建图形用户界面的工具集，尽管它包含一个画布元素，但它却没有提供在画布上绘图的函数。因此，我们需要一个额外的库来进行绘图操作。在这里，我们选择使用 **OpenGL** 作为这个额外的绘图库。

为了使用 **OpenGL**，我们需要包含 *GL/gl.h*，*windows.h*[9]和 *iupgl.h* 这几个头文件。我们也需要连接一些库，Windows 下是 *opengl32.lib*，Linux 下我们要使用 *-lGL* 连接 OpenGL 库。**IUP** 是通过 *IupGLCanvas* 控件来使用 **OpenGL** 的。为了使用 **OpenGL**，我们必须在调用 *IupOpen* 函数后，调用 *IupGLCanvasOpen* 函数。

在 *read_file* 和 *new_file* 函数中，我们调用 *imImageGetOpenGLData* 函数把图像转换为 **OpenGL** 接受的格式。我们为画布创建了一个回调，当画布需要重绘时，这个回调就会被调用。在这个回调中，我们调用 *IupGLMakeCurrent* 函数来声明这个画布是当前进行 OpenGL 绘制的画布。接着，我们设置图像对齐为 1, 调整 **OpenGL** 的默认坐标系范围为 0 到画布大小，这样就使图像的像素和窗口上的图像一一对应。我们使用 *glClearColor* 函数和 *glClear* 函数来清除画布背景。接着，通过 *GLDATA* 属性，我们得到我们需要绘制的图像数据，使用 *glDrawPixels* 函数进行绘制。需要注意，由于 **OpenGL** 的限制，*glRasterPos2i* 和 *glDrawPixels* 函数不能接受超过屏幕大小的值。需要显示的图像大小必须小于画布的大小，否则它就不会被绘制。我们可以通过使用纹理来突破这个限制。但是，这超出了我们本章的内容，所以就不详细描述了。

在本例中，我们使用了一种叫做**双缓冲**的技术。为了使用它，我们需要设置画布的 *BUFFER* 属性为 *DOUBLE*。之后，我们使用绘图函数就会绘制在后台的画布上，然后，调用 *IupGLSwapBuffers* 函数就会交换前台和后台，显示出绘制的图像。

在本例中，我们还使用了 *IupColorDlg* 这个预定义的对话框。*IupColorDlg* 会给用户显示一个调色板，供用户选择颜色。在本例中，我们使用它来改变画布的背景颜色。

在 Lua 形式下，我们通过 **LuaGL** 来使用 **OpenGL**。我们只需要包含 *luagl* 包就可以使用它。使用 *IupGLCanvas* 我们还需要包含 *iupluagl* 这个包。Lua 形式下的 *gl.Func* 等价于 C 语言形式下的 *glFunc*。

C语言

```
#ifdef WIN32
```

[9]如果在 Windows 系统下。

```
#include <windows.h>    /* necessary because of the Microsoft OpenGL headers depend
#endif
#include <GL/gl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <iup.h>
#include <iup_config.h>
#include <iupgl.h>
#include <im.h>
#include <im_image.h>
#include <im_convert.h>
#include <iupim.h>


/********************************* Utilities *************************************


const char* str_filetitle(const char *filename)
{
  /* Start at the last character */
  int len = (int)strlen(filename);
  int offset = len - 1;
  while (offset != 0)
  {
    if (filename[offset] == '\\' || filename[offset] == '/')
    {
      offset++;
      break;
    }
    offset--;
  }
  return filename + offset;
}

const char* str_fileext(const char *filename)
{
  /* Start at the last character */
  int len = (int)strlen(filename);
  int offset = len - 1;
  while (offset != 0)
```

```
  {
    if (filename[offset] == '\\' || filename[offset] == '/')
      break;

    if (filename[offset] == '.')
    {
      offset++;
      return filename + offset;
    }
    offset--;
  }
  return NULL;
}

int str_compare(const char *l, const char *r, int casesensitive)
{
  if (!l || !r)
    return 0;

  while (*l && *r)
  {
    int diff;
    char l_char = *l,
      r_char = *r;

    /* compute the difference of both characters */
    if (casesensitive)
      diff = l_char - r_char;
    else
      diff = tolower((int)l_char) - tolower((int)r_char);

    /* if they differ we have a result */
    if (diff != 0)
      return 0;

    /* otherwise process the next characters */
    ++l;
    ++r;
  }

  /* check also for terminator */
  if (*l == *r)
```

```
    return 1;

  if (*r == 0)
    return 1;  /* if second string is at terminator, then it is partially equal */

  return 0;
}

void show_error(const char* message, int is_error)
{
  Ihandle* dlg = IupMessageDlg();
  IupSetStrAttribute(dlg, "PARENTDIALOG", IupGetGlobal("PARENTDIALOG"));
  IupSetAttribute(dlg, "DIALOGTYPE", is_error ? "ERROR" : "WARNING");
  IupSetAttribute(dlg, "BUTTONS", "OK");
  IupSetStrAttribute(dlg, "TITLE", is_error ? "Error" : "Warning");
  IupSetStrAttribute(dlg, "VALUE", message);
  IupPopup(dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
  IupDestroy(dlg);
}

void show_file_error(int error)
{
  switch (error)
  {
  case IM_ERR_OPEN:
    show_error("Error␣Opening␣File.", 1);
    break;
  case IM_ERR_MEM:
    show_error("Insufficient␣memory.", 1);
    break;
  case IM_ERR_ACCESS:
    show_error("Error␣Accessing␣File.", 1);
    break;
  case IM_ERR_DATA:
    show_error("Image␣type␣not␣Supported.", 1);
    break;
  case IM_ERR_FORMAT:
    show_error("Invalid␣Format.", 1);
    break;
  case IM_ERR_COMPRESS:
    show_error("Invalid␣or␣unsupported␣compression.", 1);
    break;
```

```
  default:
    show_error("Unknown Error.", 1);
  }
}

imImage* read_file(const char* filename)
{
  int error;
  imImage* image = imFileImageLoadBitmap(filename, 0, &error);
  if (error)
    show_file_error(error);
  else
  {
    /* we are going to support only RGB images with no alpha */
    imImageRemoveAlpha(image);
    if (image->color_space != IM_RGB)
    {
      imImage* new_image = imImageCreateBased(image, -1, -1, IM_RGB, -1);
      imConvertColorSpace(image, new_image);
      imImageDestroy(image);

      image = new_image;
    }

    /* create OpenGL compatible data */
    imImageGetOpenGLData(image, NULL);
  }
  return image;
}

int write_file(const char* filename, const imImage* image)
{
  const char* format = imImageGetAttribString(image, "FileFormat");
  int error = imFileImageSave(filename, format, image);
  if (error)
  {
    show_file_error(error);
    return 0;
  }
  return 1;
}
```

```
void new_file(Ihandle* ih, imImage* image)
{
  Ihandle* dlg = IupGetDialog(ih);
  Ihandle* canvas = IupGetDialogChild(dlg, "CANVAS");
  imImage* old_image = (imImage*)IupGetAttribute(canvas, "IMAGE");

  IupSetAttribute(dlg, "TITLE", "Untitled - Simple Paint");
  IupSetAttribute(canvas, "FILENAME", NULL);
  IupSetAttribute(canvas, "DIRTY", "NO");

  IupSetAttribute(canvas, "IMAGE", (char*)image);

  /* create OpenGL compatible data */
  imImageGetOpenGLData(image, NULL);

  IupUpdate(canvas);

  if (old_image)
    imImageDestroy(old_image);
}

void check_new_file(Ihandle* dlg)
{
  Ihandle* canvas = IupGetDialogChild(dlg, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  if (!image)
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
    int width = IupConfigGetVariableIntDef(config, "NewImage", "Width", 640);
    int height = IupConfigGetVariableIntDef(config, "NewImage", "Height", 480);

    image = imImageCreate(width, height, IM_RGB, IM_BYTE);

    new_file(dlg, image);
  }
}

void open_file(Ihandle* ih, const char* filename)
{
  imImage* image = read_file(filename);
  if (image)
  {
```

```
    Ihandle* dlg = IupGetDialog(ih);
    Ihandle* canvas = IupGetDialogChild(dlg, "CANVAS");
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
    imImage* old_image = (imImage*)IupGetAttribute(canvas, "IMAGE");

    IupSetfAttribute(dlg, "TITLE", "%s - Simple Paint", str_filetitle(filename));
    IupSetStrAttribute(canvas, "FILENAME", filename);
    IupSetAttribute(canvas, "DIRTY", "NO");
    IupSetAttribute(canvas, "IMAGE", (char*)image);

    IupUpdate(canvas);

    if (old_image)
      imImageDestroy(old_image);

    IupConfigRecentUpdate(config, filename);
  }
}

void save_file(Ihandle* canvas)
{
  char* filename = IupGetAttribute(canvas, "FILENAME");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  if (write_file(filename, image))
    IupSetAttribute(canvas, "DIRTY", "NO");
}

void set_file_format(imImage* image, const char* filename)
{
  const char* ext = str_fileext(filename);
  const char* format = "JPEG";
  if (str_compare(ext, "jpg", 0) || str_compare(ext, "jpeg", 0))
    format = "JPEG";
  else if (str_compare(ext, "bmp", 0))
    format = "BMP";
  else if (str_compare(ext, "png", 0))
    format = "PNG";
  else if (str_compare(ext, "tga", 0))
    format = "TGA";
  else if (str_compare(ext, "tif", 0) || str_compare(ext, "tiff", 0))
    format = "TIFF";
  imImageSetAttribString(image, "FileFormat", format);
```

```
}

void saveas_file(Ihandle* canvas, const char* filename)
{
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");

  set_file_format(image, filename);

  if (write_file(filename, image))
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");

    IupSetfAttribute(IupGetDialog(canvas), "TITLE", "%s - Simple Paint", str_fileti
    IupSetStrAttribute(canvas, "FILENAME", filename);
    IupSetAttribute(canvas, "DIRTY", "NO");

    IupConfigRecentUpdate(config, filename);
  }
}

int save_check(Ihandle* ih)
{
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  if (IupGetInt(canvas, "DIRTY"))
  {
    switch (IupAlarm("Warning", "File not saved! Save it now?", "Yes", "No", "Cance
    {
    case 1:  /* save the changes and continue */
      save_file(canvas);
      break;
    case 2:  /* ignore the changes and continue */
      break;
    case 3:  /* cancel */
      return 0;
    }
  }
  return 1;
}

void toggle_bar_visibility(Ihandle* item, Ihandle* ih)
{
  if (IupGetInt(item, "VALUE"))
```

```
  {
    IupSetAttribute(ih, "FLOATING", "YES");
    IupSetAttribute(ih, "VISIBLE", "NO");
    IupSetAttribute(item, "VALUE", "OFF");
  }
  else
  {
    IupSetAttribute(ih, "FLOATING", "NO");
    IupSetAttribute(ih, "VISIBLE", "YES");
    IupSetAttribute(item, "VALUE", "ON");
  }

  IupRefresh(ih);  /* refresh the dialog layout */
}


/********************************* Callbacks *************************************


int canvas_action_cb(Ihandle* canvas)
{
  int x, y, canvas_width, canvas_height;
  void* gldata;
  unsigned int ri, gi, bi;
  imImage* image;
  Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
  const char* background = IupConfigGetVariableStrDef(config, "Canvas", "Background

  IupGetIntInt(canvas, "DRAWSIZE", &canvas_width, &canvas_height);

  IupGLMakeCurrent(canvas);

  /* OpenGL configuration */
  glPixelStorei(GL_UNPACK_ALIGNMENT, 1);          /* image data alignment is 1 */

  glViewport(0, 0, canvas_width, canvas_height);

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(0, canvas_width, 0, canvas_height, -1, 1);

  glMatrixMode(GL_MODELVIEW);
```

```
  glLoadIdentity();

  /* draw the background */
  sscanf(background, "%u %u %u", &ri, &gi, &bi);
  glClearColor(ri / 255.f, gi / 255.f, bi / 255.f, 1);
  glClear(GL_COLOR_BUFFER_BIT);

  /* draw the image at the center of the canvas */
  image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  if (image)
  {
    x = (canvas_width - image->width) / 2;
    y = (canvas_height - image->height) / 2;
    gldata = (void*)imImageGetAttribute(image, "GLDATA", NULL, NULL);
    glRasterPos2i(x, y);  /* this will not work for negative values, OpenGL limitat
    glDrawPixels(image->width, image->height, GL_RGB, GL_UNSIGNED_BYTE, gldata);
/* no zoom support, must use texture */
  }

  IupGLSwapBuffers(canvas);
  return IUP_DEFAULT;
}

int dropfiles_cb(Ihandle* ih, const char* filename)
{
  if (save_check(ih))
    open_file(ih, filename);

  return IUP_DEFAULT;
}

int file_menu_open_cb(Ihandle* ih)
{
  Ihandle* item_revert = IupGetDialogChild(ih, "ITEM_REVERT");
  Ihandle* item_save = IupGetDialogChild(ih, "ITEM_SAVE");
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  char* filename = IupGetAttribute(canvas, "FILENAME");
  int dirty = IupGetInt(canvas, "DIRTY");

  if (dirty)
    IupSetAttribute(item_save, "ACTIVE", "YES");
  else
```

```
    IupSetAttribute(item_save, "ACTIVE", "NO");

  if (dirty && filename)
    IupSetAttribute(item_revert, "ACTIVE", "YES");
  else
    IupSetAttribute(item_revert, "ACTIVE", "NO");
  return IUP_DEFAULT;
}

int edit_menu_open_cb(Ihandle* ih)
{
  Ihandle *clipboard = IupClipboard();

  Ihandle *item_paste = IupGetDialogChild(ih, "ITEM_PASTE");

  if (!IupGetInt(clipboard, "IMAGEAVAILABLE"))
    IupSetAttribute(item_paste, "ACTIVE", "NO");
  else
    IupSetAttribute(item_paste, "ACTIVE", "YES");

  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int config_recent_cb(Ihandle* ih)
{
  if (save_check(ih))
  {
    char* filename = IupGetAttribute(ih, "TITLE");
    open_file(ih, filename);
  }
  return IUP_DEFAULT;
}

int item_new_action_cb(Ihandle* item_new)
{
  if (save_check(item_new))
  {
    Ihandle* canvas = IupGetDialogChild(item_new, "CANVAS");
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
    int width = IupConfigGetVariableIntDef(config, "NewImage", "Width", 640);
    int height = IupConfigGetVariableIntDef(config, "NewImage", "Height", 480);
```

```
    if (IupGetParam("New Image", NULL, NULL, "Width: %i[1,]\nHeight: %i[1,]\n", &wi
    {
      imImage* image = imImageCreate(width, height, IM_RGB, IM_BYTE);

      IupConfigSetVariableInt(config, "NewImage", "Width", width);
      IupConfigSetVariableInt(config, "NewImage", "Height", height);

      new_file(item_new, image);
    }
  }

  return IUP_DEFAULT;
}

int select_file(Ihandle* parent_dlg, int is_open)
{
  Ihandle* config = (Ihandle*)IupGetAttribute(parent_dlg, "CONFIG");
  Ihandle* canvas = IupGetDialogChild(parent_dlg, "CANVAS");
  const char* dir = IupConfigGetVariableStr(config, "MainWindow", "LastDirectory");

  Ihandle* filedlg = IupFileDlg();
  if (is_open)
    IupSetAttribute(filedlg, "DIALOGTYPE", "OPEN");
  else
  {
    IupSetAttribute(filedlg, "DIALOGTYPE", "SAVE");
    IupSetStrAttribute(filedlg, "FILE", IupGetAttribute(canvas, "FILENAME"));
  }
  IupSetAttribute(filedlg, "EXTFILTER", "Image Files|*.bmp;*.jpg;*.png;*.tif;*.tga|
  IupSetStrAttribute(filedlg, "DIRECTORY", dir);
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", parent_dlg);

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    if (is_open)
      open_file(parent_dlg, filename);
    else
      saveas_file(canvas, filename);
```

```
    dir = IupGetAttribute(filedlg, "DIRECTORY");
    IupConfigSetVariableStr(config, "MainWindow", "LastDirectory", dir);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_open_action_cb(Ihandle* item_open)
{
  if (!save_check(item_open))
    return IUP_DEFAULT;

  return select_file(IupGetDialog(item_open), 1);
}

int item_saveas_action_cb(Ihandle* item_saveas)
{
  return select_file(IupGetDialog(item_saveas), 0);
}

int item_save_action_cb(Ihandle* item_save)
{
  Ihandle* canvas = IupGetDialogChild(item_save, "CANVAS");
  char* filename = IupGetAttribute(canvas, "FILENAME");
  if (!filename)
    item_saveas_action_cb(item_save);
  else
  {
    /* test again because in can be called using the hot key */
    int dirty = IupGetInt(canvas, "DIRTY");
    if (dirty)
      save_file(canvas);
  }
  return IUP_DEFAULT;
}

int item_revert_action_cb(Ihandle* item_revert)
{
  Ihandle* canvas = IupGetDialogChild(item_revert, "CANVAS");
  char* filename = IupGetAttribute(canvas, "FILENAME");
  open_file(item_revert, filename);
```

```
  return IUP_DEFAULT;
}

int item_exit_action_cb(Ihandle* item_exit)
{
  Ihandle* dlg = IupGetDialog(item_exit);
  Ihandle* config = (Ihandle*)IupGetAttribute(dlg, "CONFIG");
  Ihandle* canvas = IupGetDialogChild(dlg, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");

  if (!save_check(item_exit))
    return IUP_IGNORE;  /* to abort the CLOSE_CB callback */

  if (image)
    imImageDestroy(image);

  IupConfigDialogClosed(config, dlg, "MainWindow");
  IupConfigSave(config);
  IupDestroy(config);
  return IUP_CLOSE;
}

int item_copy_action_cb(Ihandle* item_copy)
{
  Ihandle* canvas = IupGetDialogChild(item_copy, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(clipboard, "NATIVEIMAGE", (char*)IupGetImageNativeHandle(image));
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_paste_action_cb(Ihandle* item_paste)
{
  if (save_check(item_paste))
  {
    Ihandle* canvas = IupGetDialogChild(item_paste, "CANVAS");
    imImage* old_image = (imImage*)IupGetAttribute(canvas, "IMAGE");

    Ihandle *clipboard = IupClipboard();
    imImage* image = IupGetNativeHandleImage(IupGetAttribute(clipboard, "NATIVEIMAG
    IupDestroy(clipboard);
```

```
    if (!image)
    {
      show_error("Invalid Clipboard Data", 1);
      return IUP_DEFAULT;
    }

    /* we are going to support only RGB images with no alpha */
    imImageRemoveAlpha(image);
    if (image->color_space != IM_RGB)
    {
      imImage* new_image = imImageCreateBased(image, -1, -1, IM_RGB, -1);
      imConvertColorSpace(image, new_image);
      imImageDestroy(image);

      image = new_image;
    }

    /* create OpenGL compatible data */
    imImageGetOpenGLData(image, NULL);

    imImageSetAttribString(image, "FileFormat", "JPEG");

    IupSetAttribute(canvas, "DIRTY", "Yes");
    IupSetAttribute(canvas, "IMAGE", (char*)image);
    IupSetAttribute(canvas, "FILENAME", NULL);
    IupSetAttribute(IupGetDialog(canvas), "TITLE", "Untitled - Simple Paint");

    IupUpdate(canvas);

    if (old_image)
      imImageDestroy(old_image);
  }
  return IUP_DEFAULT;
}

int item_background_action_cb(Ihandle* item_background)
{
  Ihandle* canvas = IupGetDialogChild(item_background, "CANVAS");
  Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
  Ihandle* colordlg = IupColorDlg();
  const char* background = IupConfigGetVariableStrDef(config, "Canvas", "Background
```

```
  IupSetStrAttribute(colordlg, "VALUE", background);
  IupSetAttributeHandle(colordlg, "PARENTDIALOG", IupGetDialog(item_background));

  IupPopup(colordlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(colordlg, "STATUS") == 1)
  {
    background = IupGetAttribute(colordlg, "VALUE");
    IupConfigSetVariableStr(config, "Canvas", "Background", background);

    IupUpdate(canvas);
  }

  IupDestroy(colordlg);
  return IUP_DEFAULT;
}

int item_toolbar_action_cb(Ihandle* item_toolbar)
{
  Ihandle* canvas = IupGetDialogChild(item_toolbar, "CANVAS");
  Ihandle* toolbar = IupGetChild(IupGetParent(canvas), 0);
  Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");

  toggle_bar_visibility(item_toolbar, toolbar);

  IupConfigSetVariableStr(config, "MainWindow", "Toolbar", IupGetAttribute(item_too
  return IUP_DEFAULT;
}

int item_statusbar_action_cb(Ihandle* item_statusbar)
{
  Ihandle* canvas = IupGetDialogChild(item_statusbar, "CANVAS");
  Ihandle* statusbar = IupGetBrother(canvas);
  Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");

  toggle_bar_visibility(item_statusbar, statusbar);

  IupConfigSetVariableStr(config, "MainWindow", "Statusbar", IupGetAttribute(item_s
  return IUP_DEFAULT;
}

int item_help_action_cb(void)
```

```
{
  IupHelp("http://www.tecgraf.puc-rio.br/iup");
  return IUP_DEFAULT;
}

int item_about_action_cb(void)
{
  IupMessage("About", "   Simple Paint\n\nAutors:\n   Gustavo Lyrio\n   Antonio Scu
  return IUP_DEFAULT;
}


/********************************* Main ****************************************/


Ihandle* create_main_dialog(Ihandle *config)
{
  Ihandle *dlg, *vbox, *canvas, *menu;
  Ihandle *sub_menu_file, *file_menu, *item_exit, *item_new, *item_open, *item_save
  Ihandle *sub_menu_edit, *edit_menu, *item_copy, *item_paste;
  Ihandle *btn_copy, *btn_paste, *btn_new, *btn_open, *btn_save;
  Ihandle *sub_menu_help, *help_menu, *item_help, *item_about;
  Ihandle *sub_menu_view, *view_menu, *item_toolbar, *item_statusbar;
  Ihandle *statusbar, *toolbar, *recent_menu, *item_background;

  canvas = IupGLCanvas(NULL);
  IupSetAttribute(canvas, "NAME", "CANVAS");
  IupSetAttribute(canvas, "DIRTY", "NO");
  IupSetAttribute(canvas, "BUFFER", "DOUBLE");
  IupSetCallback(canvas, "ACTION", (Icallback)canvas_action_cb);
  IupSetCallback(canvas, "DROPFILES_CB", (Icallback)dropfiles_cb);

  statusbar = IupLabel("(0, 0) = [0   0   0]");
  IupSetAttribute(statusbar, "NAME", "STATUSBAR");
  IupSetAttribute(statusbar, "EXPAND", "HORIZONTAL");
  IupSetAttribute(statusbar, "PADDING", "10x5");

  item_new = IupItem("&New\tCtrl+N", NULL);
  IupSetAttribute(item_new, "IMAGE", "IUP_FileNew");
  IupSetCallback(item_new, "ACTION", (Icallback)item_new_action_cb);
  btn_new = IupButton(NULL, NULL);
  IupSetAttribute(btn_new, "IMAGE", "IUP_FileNew");
```

```
IupSetAttribute(btn_new, "FLAT", "Yes");
IupSetCallback(btn_new, "ACTION", (Icallback)item_new_action_cb);
IupSetAttribute(btn_new, "TIP", "New (Ctrl+N)");
IupSetAttribute(btn_new, "CANFOCUS", "No");

item_open = IupItem("&Open...\tCtrl+O", NULL);
IupSetAttribute(item_open, "IMAGE", "IUP_FileOpen");
IupSetCallback(item_open, "ACTION", (Icallback)item_open_action_cb);
btn_open = IupButton(NULL, NULL);
IupSetAttribute(btn_open, "IMAGE", "IUP_FileOpen");
IupSetAttribute(btn_open, "FLAT", "Yes");
IupSetCallback(btn_open, "ACTION", (Icallback)item_open_action_cb);
IupSetAttribute(btn_open, "TIP", "Open (Ctrl+O)");
IupSetAttribute(btn_open, "CANFOCUS", "No");

item_save = IupItem("&Save\tCtrl+S", NULL);
IupSetAttribute(item_save, "NAME", "ITEM_SAVE");
IupSetAttribute(item_save, "IMAGE", "IUP_FileSave");
IupSetCallback(item_save, "ACTION", (Icallback)item_save_action_cb);
btn_save = IupButton(NULL, NULL);
IupSetAttribute(btn_save, "IMAGE", "IUP_FileSave");
IupSetAttribute(btn_save, "FLAT", "Yes");
IupSetCallback(btn_save, "ACTION", (Icallback)item_save_action_cb);
IupSetAttribute(btn_save, "TIP", "Save (Ctrl+S)");
IupSetAttribute(btn_save, "CANFOCUS", "No");

item_saveas = IupItem("Save &As...", NULL);
IupSetAttribute(item_saveas, "NAME", "ITEM_SAVEAS");
IupSetCallback(item_saveas, "ACTION", (Icallback)item_saveas_action_cb);

item_revert = IupItem("&Revert", NULL);
IupSetAttribute(item_revert, "NAME", "ITEM_REVERT");
IupSetCallback(item_revert, "ACTION", (Icallback)item_revert_action_cb);

item_exit = IupItem("E&xit", NULL);
IupSetCallback(item_exit, "ACTION", (Icallback)item_exit_action_cb);

item_copy = IupItem("&Copy\tCtrl+C", NULL);
IupSetAttribute(item_copy, "NAME", "ITEM_COPY");
IupSetAttribute(item_copy, "IMAGE", "IUP_EditCopy");
IupSetCallback(item_copy, "ACTION", (Icallback)item_copy_action_cb);
btn_copy = IupButton(NULL, NULL);
```

```
  IupSetAttribute(btn_copy, "IMAGE", "IUP_EditCopy");
  IupSetAttribute(btn_copy, "FLAT", "Yes");
  IupSetCallback(btn_copy, "ACTION", (Icallback)item_copy_action_cb);
  IupSetAttribute(btn_copy, "TIP", "Copy (Ctrl+C)");
  IupSetAttribute(btn_copy, "CANFOCUS", "No");

  item_paste = IupItem("&Paste\tCtrl+V", NULL);
  IupSetAttribute(item_paste, "NAME", "ITEM_PASTE");
  IupSetAttribute(item_paste, "IMAGE", "IUP_EditPaste");
  IupSetCallback(item_paste, "ACTION", (Icallback)item_paste_action_cb);
  btn_paste = IupButton(NULL, NULL);
  IupSetAttribute(btn_paste, "IMAGE", "IUP_EditPaste");
  IupSetAttribute(btn_paste, "FLAT", "Yes");
  IupSetCallback(btn_paste, "ACTION", (Icallback)item_paste_action_cb);
  IupSetAttribute(btn_paste, "TIP", "Paste (Ctrl+V)");
  IupSetAttribute(btn_paste, "CANFOCUS", "No");

  item_background = IupItem("&Background...", NULL);
  IupSetCallback(item_background, "ACTION", (Icallback)item_background_action_cb);

  item_toolbar = IupItem("&Toobar", NULL);
  IupSetCallback(item_toolbar, "ACTION", (Icallback)item_toolbar_action_cb);
  IupSetAttribute(item_toolbar, "VALUE", "ON");

  item_statusbar = IupItem("&Statusbar", NULL);
  IupSetCallback(item_statusbar, "ACTION", (Icallback)item_statusbar_action_cb);
  IupSetAttribute(item_statusbar, "VALUE", "ON");

  item_help = IupItem("&Help...", NULL);
  IupSetCallback(item_help, "ACTION", (Icallback)item_help_action_cb);

  item_about = IupItem("&About...", NULL);
  IupSetCallback(item_about, "ACTION", (Icallback)item_about_action_cb);

  recent_menu = IupMenu(NULL);

  file_menu = IupMenu(
    item_new,
    item_open,
    item_save,
    item_saveas,
    item_revert,
```

```
    IupSeparator(),
    IupSubmenu("Recent &Files", recent_menu),
    item_exit,
    NULL);
  edit_menu = IupMenu(
    item_copy,
    item_paste,
    NULL);
  view_menu = IupMenu(
    item_background,
    IupSeparator(),
    item_toolbar,
    item_statusbar,
    NULL);
  help_menu = IupMenu(
    item_help,
    item_about,
    NULL);

  IupSetCallback(file_menu, "OPEN_CB", (Icallback)file_menu_open_cb);
  IupSetCallback(edit_menu, "OPEN_CB", (Icallback)edit_menu_open_cb);

  sub_menu_file = IupSubmenu("&File", file_menu);
  sub_menu_edit = IupSubmenu("&Edit", edit_menu);
  sub_menu_view = IupSubmenu("&View", view_menu);
  sub_menu_help = IupSubmenu("&Help", help_menu);

  menu = IupMenu(
    sub_menu_file,
    sub_menu_edit,
    sub_menu_view,
    sub_menu_help,
    NULL);

  toolbar = IupHbox(
    btn_new,
    btn_open,
    btn_save,
    IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
    btn_copy,
    btn_paste,
    NULL);
```

```
  IupSetAttribute(toolbar, "MARGIN", "5x5");
  IupSetAttribute(toolbar, "GAP", "2");

  vbox = IupVbox(
    toolbar,
    canvas,
    statusbar,
    NULL);

  dlg = IupDialog(vbox);
  IupSetAttributeHandle(dlg, "MENU", menu);
  IupSetAttribute(dlg, "SIZE", "HALFxHALF");
  IupSetCallback(dlg, "CLOSE_CB", (Icallback)item_exit_action_cb);
  IupSetCallback(dlg, "DROPFILES_CB", (Icallback)dropfiles_cb);

  IupSetCallback(dlg, "K_cN", (Icallback)item_new_action_cb);
  IupSetCallback(dlg, "K_cO", (Icallback)item_open_action_cb);
  IupSetCallback(dlg, "K_cS", (Icallback)item_save_action_cb);
  IupSetCallback(dlg, "K_cV", (Icallback)item_paste_action_cb);
  IupSetCallback(dlg, "K_cC", (Icallback)item_copy_action_cb);

  /* parent for pre-defined dialogs in closed functions (IupMessage and IupAlarm) *
  IupSetAttributeHandle(NULL, "PARENTDIALOG", dlg);

  /* Initialize variables from the configuration file */

  IupConfigRecentInit(config, recent_menu, config_recent_cb, 10);

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Toolbar", 1))
  {
    IupSetAttribute(item_toolbar, "VALUE", "OFF");

    IupSetAttribute(toolbar, "FLOATING", "YES");
    IupSetAttribute(toolbar, "VISIBLE", "NO");
  }

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Statusbar", 1))
  {
    IupSetAttribute(item_statusbar, "VALUE", "OFF");

    IupSetAttribute(statusbar, "FLOATING", "YES");
    IupSetAttribute(statusbar, "VISIBLE", "NO");
```

```
  }

  IupSetAttribute(dlg, "CONFIG", (char*)config);

  return dlg;
}

int main(int argc, char **argv)
{
  Ihandle *dlg;
  Ihandle *config;

  IupOpen(&argc, &argv);
  IupGLCanvasOpen();
  IupImageLibOpen();

  config = IupConfig();
  IupSetAttribute(config, "APP_NAME", "simple_paint");
  IupConfigLoad(config);

  dlg = create_main_dialog(config);

  /* show the dialog at the last position, with the last size */
  IupConfigDialogShow(config, dlg, "MainWindow");

  /* open a file from the command line (allow file association in Windows) */
  if (argc > 1 && argv[1])
  {
    const char* filename = argv[1];
    open_file(dlg, filename);
  }

  /* initialize the current file, if not already loaded */
  check_new_file(dlg);

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")
require("iupluaimglib")
require("imlua")
require("iupluaim")
require("iupluagl")
require("luagl")


--********************************** Utilities ******************************************

function str_splitfilename(filename)
  return string.match(filename, "(.-)([^\\/]-%.?([^%.\\/]*))$")
end

function str_fileext(filename)
  local path, title, ext = str_splitfilename(filename)
  return ext
end

function str_filetitle(filename)
  local path, title, ext = str_splitfilename(filename)
  return title
end

function show_error(message, is_error)
  local dlg = iup.messagedlg{
    parentdialog = iup.GetGlobal("PARENTDIALOG"),
    buttons = "OK",
    value = message,
  }
  if (is_error) then
    dlg.dialogtype = "ERROR"
    dlg.title = "Error"
  else
    dlg.dialogtype = "WARNING"
    dlg.title = "Warning"
  end
  dlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)
  dlg:destroy()
end

function read_file(filename)
```

```
  local image, err = im.FileImageLoadBitmap(filename, 0)
  if (err) then
    show_error(im.ErrorStr(err), true)
  else
    -- we are going to support only RGB images with no alpha
    image:RemoveAlpha()
    if (image:ColorSpace() ~= im.RGB) then
      local new_image = im.ImageCreateBased(image, nil, nil, im.RGB, nil)

      im.ConvertColorSpace(image, new_image)
      image:Destroy()

      image = new_image
    end

    -- create OpenGL compatible data
    local gldata = image:GetOpenGLData()
    canvas.gldata = gldata
  end
  return image
end

function write_file(filename, image)
  local format = image:GetAttribString("FileFormat")
  local err = im.FileImageSave(filename, format, image)
  if (err and err ~= im.ERR_NONE) then
    show_error(im.ErrorStr(err), true)
    return false
  end
  return true
end

function new_file(ih, image)
  local dlg = iup.GetDialog(ih)
  local canvas = dlg.canvas
  local old_image = canvas.image

  dlg.title = "Untitled - Simple Paint"
  canvas.filename = nil
  canvas.dirty = nil
  canvas.image = image
```

```
  -- create OpenGL compatible data
  local gldata = image:GetOpenGLData()
  canvas.gldata = gldata

  iup.Update(canvas)

  if (old_image) then
    old_image:Destroy()
  end
end

function check_new_file(dlg)
  local canvas = dlg.canvas
  local image = canvas.image
  if (not image) then
    local config = canvas.config
    local width = config:GetVariableDef("NewImage", "Width", 640)
    local height = config:GetVariableDef("NewImage", "Height", 480)

    local image = im.ImageCreate(width, height, im.RGB, im.BYTE)

    new_file(dlg, image)
  end
end

function open_file(ih, filename)
  local image = read_file(filename)
  if (image) then
    local dlg = iup.GetDialog(ih)
    local canvas = dlg.canvas
    local config = canvas.config
    local old_image = canvas.image

    dlg.title = str_filetitle(filename).." - Simple Paint"
    canvas.filename = filename
    canvas.dirty = nil
    canvas.image = image

    iup.Update(canvas)

    if (old_image) then
      old_image:Destroy()
```

```
    end

    config:RecentUpdate(filename)
  end
end

function save_file(canvas)
  if (write_file(canvas.filename, canvas.image)) then
    canvas.dirty = nil
  end
end

function set_file_format(image, filename)
  local ext = str_fileext(filename)
  ext:lower()
  local format = "JPEG"
  if (ext == "jpg" or ext == "jpeg") then
    format = "JPEG"
  elseif (ext == "bmp") then
    format = "BMP"
  elseif (ext == "png") then
    format = "PNG"
  elseif (ext == "tga") then
    format = "TGA"
  elseif (ext == "tif" or ext == "tiff") then
    format = "TIFF"
  end
  image:SetAttribString("FileFormat", format)
end

function saveas_file(canvas, filename)
  local image = canvas.image

  set_file_format(image, filename)

  if (write_file(filename, image)) then
    local dlg = iup.GetDialog(canvas)
    local config = canvas.config

    dlg.title = str_filetitle(filename).." - Simple Paint"
    canvas.filename = filename
    canvas.dirty = nil
```

```
    config:RecentUpdate(filename)
  end
end

function save_check(ih)
  local dlg = iup.GetDialog(ih)
  local canvas = dlg.canvas

  if (canvas.dirty) then
    local resp = iup.Alarm("Warning", "File not saved! Save it now?", "Yes", "No",
    if resp == 1 then -- save the changes and continue
      save_file(canvas)
    elseif resp == 3 then  -- cancel
      return false
    else  -- ignore the changes and continue
    end
  end
  return true
end

function toggle_bar_visibility(item, bar)
  if (item.value == "ON") then
    bar.floating = "YES"
    bar.visible = "NO"
    item.value = "OFF"
  else
    bar.floating = "NO"
    bar.visible = "YES"
    item.value = "ON"
  end
  iup.Refresh(bar)  -- refresh the dialog layout
end


--********************************* Main (Part 1/2) ****************************

-- create all the elements that will have callbacks in Lua prior to callbacks defin

config = iup.config{}
config.app_name = "simple_paint"
config:Load()
```

```
statusbar = iup.label{title = "(0, 0) = [0 0 0]", expand = "HORIZONTAL", paddin

canvas = iup.glcanvas{
  config = config,
  dirty = nil,
  buffer = "DOUBLE",
}

item_new = iup.item{title = "&New...\tCtrl+N", image = "IUP_FileNew"}
item_open = iup.item{title = "&Open...\tCtrl+O", image = "IUP_FileOpen"}
item_save = iup.item{title="&Save\tCtrl+S"}
item_saveas = iup.item{title="Save &As...", image = "IUP_FileSave"}
item_revert = iup.item{title="&Revert"}
item_exit = iup.item{title="E&xit"}
item_copy = iup.item{title="&Copy\tCtrl+C", image = "IUP_EditCopy"}
item_paste = iup.item{title="&Paste\tCtrl+V", image = "IUP_EditPaste"}
item_background = iup.item{title="&Background..."}
item_toolbar = iup.item{title="&Toobar", value="ON"}
item_statusbar = iup.item{title="&Statusbar", value="ON"}
item_help = iup.item{title="&Help..."}
item_about = iup.item{title="&About..."}

recent_menu = iup.menu{}

file_menu = iup.menu{
  item_new,
  item_open,
  item_save,
  item_saveas,
  item_revert,
  iup.separator{},
  iup.submenu{title="Recent &Files", recent_menu},
  item_exit
  }

edit_menu = iup.menu{
  item_copy,
  item_paste,
  }

view_menu = iup.menu{
```

```
  item_background,
  iup.separator{},
  item_toolbar,
  item_statusbar,
  }
help_menu = iup.menu{item_help, item_about}

sub_menu_file = iup.submenu{file_menu, title = "&File"}
sub_menu_edit = iup.submenu{edit_menu, title = "&Edit"}
sub_menu_view = iup.submenu{title = "&View", view_menu}
sub_menu_help = iup.submenu{help_menu, title = "&Help"}

menu = iup.menu{
  sub_menu_file,
  sub_menu_edit,
  sub_menu_view,
  sub_menu_help,
  }


--********************************* Callbacks *********************************


function canvas:action()
  local image = canvas.image
  local canvas_width, canvas_height = string.match(canvas.drawsize,"(%d*)x(%d*)")

  canvas:MakeCurrent()

  -- OpenGL configuration
  gl.PixelStore(gl.UNPACK_ALIGNMENT, 1)          -- image data alignment is 1

  gl.Viewport(0, 0, canvas_width, canvas_height)

  gl.MatrixMode(gl.PROJECTION)
  gl.LoadIdentity()
  gl.Ortho(0, canvas_width, 0, canvas_height, -1, 1)

  gl.MatrixMode(gl.MODELVIEW)
  gl.LoadIdentity()

  -- draw the background
```

```
  local background = config:GetVariableDef("Canvas", "Background", "255 255 255")
  local r, g, b = string.match(background, "(%d*) (%d*) (%d*)")
  gl.ClearColor(r / 255, g / 255, b / 255, 1)
  gl.Clear(gl.COLOR_BUFFER_BIT)

  -- draw the image at the center of the canvas
  if (image) then
    local x = (canvas_width - image:Width()) / 2
    local y = (canvas_height - image:Height()) / 2
    gl.RasterPos(x, y)  -- this will not work for negative values, an OpenGL limita
    gl.DrawPixelsRaw(image:Width(), image:Height(), gl.RGB, gl.UNSIGNED_BYTE, canva
-- no zoom support, must use texture
  end

  canvas:SwapBuffers()
end

function canvas:dropfiles_cb(filename)
  if (save_check(self)) then
    open_file(self, filename)
  end
end

function file_menu:open_cb()
  if (canvas.dirty) then
    item_save.active = "YES"
  else
    item_save.active = "NO"
  end
  if (canvas.dirty and canvas.filename) then
    item_revert.active = "YES"
  else
    item_revert.active = "NO"
  end
end

function edit_menu:open_cb()
  local clipboard = iup.clipboard{}
  if (clipboard.imageavailable == "NO") then
    item_paste.active = "NO"
  else
    item_paste.active = "YES"
```

```
  end
  clipboard:destroy()
end

function config:recent_cb()
  if (save_check(self)) then
    local filename = self.title
    open_file(self, filename)
  end
end

function item_new:action()
  if save_check(self) then
    local width = config:GetVariableDef("NewImage", "Width", 640)
    local height = config:GetVariableDef("NewImage", "Height", 480)

    local ret, new_width, new_height = iup.GetParam("New Image", nil, "Width: %i[1,
    if (ret) then
      local new_image = im.ImageCreate(new_width, new_height, im.RGB, im.BYTE)

      config:SetVariable("NewImage", "Width", new_width)
      config:SetVariable("NewImage", "Height", new_height)

      new_file(item_new, new_image)
    end
  end
end

function select_file(parent_dlg, is_open)
  local filedlg = iup.filedlg{
    extfilter="Image Files|*.bmp;*.jpg;*.png;*.tif;*.tga|All Files|*.*|",
    parentdialog = parent_dlg,
    directory = config:GetVariable("MainWindow", "LastDirectory"),
    }

  if (is_open) then
    filedlg.dialogtype = "OPEN"
  else
    filedlg.dialogtype = "SAVE"
    filedlg.file = canvas.filename
  end
```

```
  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    if (is_open) then
      open_file(parent_dlg, filename)
    else
      saveas_file(canvas, filename)
    end

    config:SetVariable("MainWindow", "LastDirectory", filedlg.directory)
  end

  filedlg:destroy()
end

function item_open:action()
  if not save_check(self) then
    return
  end

  select_file(dlg, true)
end

function item_saveas:action()
  select_file(dlg, false)
end

function item_save:action()
  if (not canvas.filename) then
    item_saveas:action()
  else
    -- test again because in can be called using the hot key
    if (canvas.dirty) then
      save_file(canvas)
    end
  end
end

function item_revert:action()
  open_file(self, canvas.filename)
end
```

```
function item_exit:action()
  local image = canvas.image

  if not save_check(self) then
    return iup.IGNORE  -- to abort the CLOSE_CB callback
  end

  if (image) then
    image:Destroy()
  end

  config:DialogClosed(iup.GetDialog(self), "MainWindow")
  config:Save()
  config:destroy()
  return iup.CLOSE
end

function item_copy:action()
  local clipboard = iup.clipboard{}
  -- must use iup.SetAttribute because it is an userdata
  iup.SetAttribute(clipboard, "NATIVEIMAGE", iup.GetImageNativeHandle(canvas.image))
  clipboard:destroy()
end

function item_paste:action()
  if save_check(self) then
    local clipboard = iup.clipboard{}
    local old_image = canvas.image

    local image = iup.GetNativeHandleImage(clipboard.nativeimage)

    -- we are going to support only RGB images with no alpha
    image:RemoveAlpha()
    if (image:ColorSpace() ~= im.RGB) then
      local new_image = im.ImageCreateBased(image, nil, nil, im.RGB, nil)

      im.ConvertColorSpace(image, new_image)
      image:Destroy()

      image = new_image
    end
```

```
    -- create OpenGL compatible data
    local gldata = image:GetOpenGLData()
    canvas.gldata = gldata

    image:SetAttribString("FileFormat", "JPEG")

    canvas.dirty = "Yes"
    canvas.image = image
    canvas.filename = nil
    dlg.title = "Untitled - Simple Paint"

    iup.Update(canvas)

    if (old_image) then
      old_image:Destroy()
    end

    clipboard:destroy()
  end
end

function item_background:action()
  local colordlg = iup.colordlg{}
  local background = config:GetVariableDef("Canvas", "Background", "255 255 255")
  colordlg.value = background
  colordlg.parentdialog = iup.GetDialog(self)

  colordlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(colordlg.status) == 1) then
    background = colordlg.value
    config:SetVariable("Canvas", "Background", background)

    iup.Update(canvas)
  end

  colordlg:destroy()
end

function item_toolbar:action()
  toggle_bar_visibility(self, toolbar)
```

```
  config:SetVariable("MainWindow", "Toolbar", item_toolbar.value)
end

function item_statusbar:action()
  toggle_bar_visibility(self, statusbar)
  config:SetVariable("MainWindow", "Statusbar", item_statusbar.value)
end

function item_help:action()
  iup.Help("http://www.tecgraf.puc-rio.br/iup")
end

function item_about:action()
  iup.Message("About", "␣␣␣Simple␣Paint\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣Sc
end


--********************************* Main (Part 2/2) ****************************


btn_new = iup.button{image = "IUP_FileNew", flat = "Yes", action = item_new.action,
btn_open = iup.button{image = "IUP_FileOpen", flat = "Yes", action = item_open.actio
btn_save = iup.button{image = "IUP_FileSave", flat = "Yes", action = item_save.actio
btn_copy = iup.button{image =  "IUP_EditCopy", flat = "Yes", action = item_copy.act
btn_paste = iup.button{image = "IUP_EditPaste", flat = "Yes", action = item_paste.a

toolbar = iup.hbox{
  btn_new,
  btn_open,
  btn_save,
  iup.label{separator="VERTICAL"},
  btn_copy,
  btn_paste,
  margin = "5x5",
  gap = 2,
}

vbox = iup.vbox{
  toolbar,
  canvas,
  statusbar,
}
```

```
dlg = iup.dialog{
  vbox,
  title = "Simple Paint",
  size = "HALFxHALF",
  menu = menu,
  close_cb = item_exit.action,
  canvas = canvas,
  dropfiles_cb = canvas.dropfiles_cb,
}

function dlg:k_any(c)
  if (c == iup.K_cN) then
    item_new:action()
  elseif (c == iup.K_cO) then
    item_open:action()
  elseif (c == iup.K_cS) then
    item_save:action()
  elseif (c == iup.K_cV) then
    item_paste:action()
  elseif (c == iup.K_cC) then
    item_copy:action()
  end
end

-- parent for pre-defined dialogs in closed functions (IupMessage and IupAlarm)
iup.SetGlobal("PARENTDIALOG", iup.SetHandleName(dlg))

-- Initialize variables from the configuration file

config:RecentInit(recent_menu, 10)

show_statusbar = config:GetVariableDef("MainWindow", "Statusbar", "ON")
if (show_statusbar == "OFF") then
  item_statusbar.value = "OFF"
  statusbar.floating = "YES"
  statusbar.visible = "NO"
end

show_toolbar = config:GetVariableDef("MainWindow", "Toolbar", "ON")
if (show_toolbar == "OFF") then
  item_toolbar.value = "OFF"
```

```
  toolbar.floating = "YES"
  toolbar.visible = "NO"
end

-- show the dialog at the last position, with the last size
config:DialogShow(dlg, "MainWindow")

-- open a file from the command line (allow file association in Windows)
if (arg and arg[1]) then
  filename = arg[1]
  open_file(dlg, filename)
end

-- initialize the current file, if not already loaded
check_new_file(dlg)

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

4.3 使用 CD 库和打印机

在本章，我们将介绍一个 **OpenGL** 库的替代方案。尽管 **OpenGL** 的性能很好，但是它不直接支持打印，不支持 *metafile* 输出，也没有提供高质量的文本绘制功能。所以，我们需要另外的绘制工具。我们创建了一个叫做 **Canvas Draw** 的库来进行这项工作。可以在 *www.tecgraf.puc-rio.br/cd* 上找到它。设计之初，我们就是为了把它和 **IUP** 相结合。

我们需要连接 *cd* 和 *iupcd* 两个库，之后才能使用它们。C 语言还需要包含 *cd.h* 和 *cdiup.h* 两个头文件。

由于我们使用 **CD** 库来替换 **OpenGL**，所以也需要使用 *IupCanvas* 控件来替换 *IupGLCanvas* 控件。我们现在也可以移除关于 **OpenGL** 的一切。

代码里的 **OpenGL** 函数调用已经完全不需要了。*IupGLMakeCurrent* 函数被替换为 *cdCanvasActivate*，*glClearColor* 函数被替换为 *cdCanvasBackground*，*glClear* 函数被替换为 *cdCanvasClear*，*IupGLSwapBuffers* 函数被替换为 *cdCanvasFlush*。

我们添加了 *MAP_CB* 和 *UNMAP_CB* 两个回调。*MAP_CB* 的回调函数为 *canvas_map_cb* 函数，它负责使用 *cdCreateCanvas* 函数来创建 CD 画布。这一步非常重要，因为创建 CD 画布，**IUP** 画布必须先被原生系统映射。该函数接收了一个叫做 *CD_IUPDBUFFER* 的参数。这个参数说明我们使用**双缓冲**进行绘制。注意到，我们现在使用了两种类型的画布，*IupCanvas* 和 *cdCanvas*。我们通过 *IupSetAttribute* 函数来把两者关联起来。*UNMAP_CB* 的回调函数 *canvas_unmap_cb* 获取 *IupCanvas* 元素关联的 CD 画布，然后使用 *cdKillCanvas* 函数销毁它。*canvas_action_cb* 回调函数负责绘制图像。原来的一些 *OpenGL* 调用被替换为了 *imcdCanvasPutImage* 函数调用。这个函数可以在 CD 画布上绘制 **IM** 图像。

由于 CD 库支持打印功能,我们可以很方便地为我们的画图程序增加打印支持。*item_pagesetup_action_cb* 函数负责获取打印页面的信息。*view_fit_rect* 函数用于调整屏幕来显示完整的图像。*item_print_action_cb* 用于显示打印对话框。

我们添加了两个新的菜单项目：*item_pagesetup* 和 *item_print*。打印菜单项还关联了 *Ctrl+P* 热键。

在 Lua 形式下，我们需要包含 *cdlua* 和 *iupluacd* 两个包。我们可以删除之前例子中的 *luagl* 和 *iupluagl* 两个包。CD 库的函数调用的 Lua 形式为 *cd.*，也就是说 *cd.CreateCanvas* 函数等价于 C 语言的 *cdCreateCanvas* 函数。需要注意的是,*imcdCanvasPutImage* 使用 *im* 前缀，因为它是 *IM* 库的函数。

C语言

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <iup.h>
#include <iup_config.h>
#include <iupgl.h>
#include <cd.h>
#include <cdprint.h>
#include <cdiup.h>
#include <im.h>
#include <im_image.h>
#include <im_convert.h>
#include <iupim.h>


/********************************** Utilities *****************************************


const char* str_filetitle(const char *filename)
{
  /* Start at the last character */
  int len = (int)strlen(filename);
  int offset = len - 1;
  while (offset != 0)
  {
    if (filename[offset] == '\\' || filename[offset] == '/')
    {
      offset++;
      break;
    }
    offset--;
```

```
  }
  return filename + offset;
}

const char* str_fileext(const char *filename)
{
  /* Start at the last character */
  int len = (int)strlen(filename);
  int offset = len - 1;
  while (offset != 0)
  {
    if (filename[offset] == '\\' || filename[offset] == '/')
      break;

    if (filename[offset] == '.')
    {
      offset++;
      return filename + offset;
    }
    offset--;
  }
  return NULL;
}

int str_compare(const char *l, const char *r, int casesensitive)
{
  if (!l || !r)
    return 0;

  while (*l && *r)
  {
    int diff;
    char l_char = *l,
      r_char = *r;

    /* compute the difference of both characters */
    if (casesensitive)
      diff = l_char - r_char;
    else
      diff = tolower((int)l_char) - tolower((int)r_char);

    /* if they differ we have a result */
```

```
    if (diff != 0)
      return 0;

    /* otherwise process the next characters */
    ++l;
    ++r;
  }

  /* check also for terminator */
  if (*l == *r)
    return 1;

  if (*r == 0)
    return 1;  /* if second string is at terminator, then it is partially equal */

  return 0;
}

void show_error(const char* message, int is_error)
{
  Ihandle* dlg = IupMessageDlg();
  IupSetStrAttribute(dlg, "PARENTDIALOG", IupGetGlobal("PARENTDIALOG"));
  IupSetAttribute(dlg, "DIALOGTYPE", is_error ? "ERROR" : "WARNING");
  IupSetAttribute(dlg, "BUTTONS", "OK");
  IupSetStrAttribute(dlg, "TITLE", is_error ? "Error" : "Warning");
  IupSetStrAttribute(dlg, "VALUE", message);
  IupPopup(dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
  IupDestroy(dlg);
}

void show_file_error(int error)
{
  switch (error)
  {
  case IM_ERR_OPEN:
    show_error("Error␣Opening␣File.", 1);
    break;
  case IM_ERR_MEM:
    show_error("Insufficient␣memory.", 1);
    break;
  case IM_ERR_ACCESS:
    show_error("Error␣Accessing␣File.", 1);
```

```
    break;
  case IM_ERR_DATA:
    show_error("Image type not Supported.", 1);
    break;
  case IM_ERR_FORMAT:
    show_error("Invalid Format.", 1);
    break;
  case IM_ERR_COMPRESS:
    show_error("Invalid or unsupported compression.", 1);
    break;
  default:
    show_error("Unknown Error.", 1);
  }
}

imImage* read_file(const char* filename)
{
  int error;
  imImage* image = imFileImageLoadBitmap(filename, 0, &error);
  if (error)
    show_file_error(error);
  else
  {
    /* we are going to support only RGB images with no alpha */
    imImageRemoveAlpha(image);
    if (image->color_space != IM_RGB)
    {
      imImage* new_image = imImageCreateBased(image, -1, -1, IM_RGB, -1);
      imConvertColorSpace(image, new_image);
      imImageDestroy(image);

      image = new_image;
    }
  }
  return image;
}

int write_file(const char* filename, const imImage* image)
{
  const char* format = imImageGetAttribString(image, "FileFormat");
  int error = imFileImageSave(filename, format, image);
  if (error)
```

```
  {
    show_file_error(error);
    return 0;
  }
  return 1;
}

void new_file(Ihandle* ih, imImage* image)
{
  Ihandle* dlg = IupGetDialog(ih);
  Ihandle* canvas = IupGetDialogChild(dlg, "CANVAS");
  imImage* old_image = (imImage*)IupGetAttribute(canvas, "IMAGE");

  IupSetAttribute(dlg, "TITLE", "Untitled - Simple Paint");
  IupSetAttribute(canvas, "FILENAME", NULL);
  IupSetAttribute(canvas, "DIRTY", "NO");

  IupSetAttribute(canvas, "IMAGE", (char*)image);

  IupUpdate(canvas);

  if (old_image)
    imImageDestroy(old_image);
}

void check_new_file(Ihandle* dlg)
{
  Ihandle* canvas = IupGetDialogChild(dlg, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  if (!image)
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
    int width = IupConfigGetVariableIntDef(config, "NewImage", "Width", 640);
    int height = IupConfigGetVariableIntDef(config, "NewImage", "Height", 480);

    image = imImageCreate(width, height, IM_RGB, IM_BYTE);

    new_file(dlg, image);
  }
}

void open_file(Ihandle* ih, const char* filename)
```

```
{
  imImage* image = read_file(filename);
  if (image)
  {
    Ihandle* dlg = IupGetDialog(ih);
    Ihandle* canvas = IupGetDialogChild(dlg, "CANVAS");
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
    imImage* old_image = (imImage*)IupGetAttribute(canvas, "IMAGE");

    IupSetfAttribute(dlg, "TITLE", "%s - Simple Paint", str_filetitle(filename));
    IupSetStrAttribute(canvas, "FILENAME", filename);
    IupSetAttribute(canvas, "DIRTY", "NO");
    IupSetAttribute(canvas, "IMAGE", (char*)image);

    IupUpdate(canvas);

    if (old_image)
      imImageDestroy(old_image);

    IupConfigRecentUpdate(config, filename);
  }
}

void save_file(Ihandle* canvas)
{
  char* filename = IupGetAttribute(canvas, "FILENAME");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  if (write_file(filename, image))
    IupSetAttribute(canvas, "DIRTY", "NO");
}

void set_file_format(imImage* image, const char* filename)
{
  const char* ext = str_fileext(filename);
  const char* format = "JPEG";
  if (str_compare(ext, "jpg", 0) || str_compare(ext, "jpeg", 0))
    format = "JPEG";
  else if (str_compare(ext, "bmp", 0))
    format = "BMP";
  else if (str_compare(ext, "png", 0))
    format = "PNG";
  else if (str_compare(ext, "tga", 0))
```

```
    format = "TGA";
  else if (str_compare(ext, "tif", 0) || str_compare(ext, "tiff", 0))
    format = "TIFF";
  imImageSetAttribString(image, "FileFormat", format);
}

void saveas_file(Ihandle* canvas, const char* filename)
{
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");

  set_file_format(image, filename);

  if (write_file(filename, image))
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");

    IupSetfAttribute(IupGetDialog(canvas), "TITLE", "%s - Simple Paint", str_fileti
    IupSetStrAttribute(canvas, "FILENAME", filename);
    IupSetAttribute(canvas, "DIRTY", "NO");

    IupConfigRecentUpdate(config, filename);
  }
}

int save_check(Ihandle* ih)
{
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  if (IupGetInt(canvas, "DIRTY"))
  {
    switch (IupAlarm("Warning", "File not saved! Save it now?", "Yes", "No", "Cance
    {
    case 1:  /* save the changes and continue */
      save_file(canvas);
      break;
    case 2:  /* ignore the changes and continue */
      break;
    case 3:  /* cancel */
      return 0;
    }
  }
  return 1;
}
```

```
void toggle_bar_visibility(Ihandle* item, Ihandle* ih)
{
  if (IupGetInt(item, "VALUE"))
  {
    IupSetAttribute(ih, "FLOATING", "YES");
    IupSetAttribute(ih, "VISIBLE", "NO");
    IupSetAttribute(item, "VALUE", "OFF");
  }
  else
  {
    IupSetAttribute(ih, "FLOATING", "NO");
    IupSetAttribute(ih, "VISIBLE", "YES");
    IupSetAttribute(item, "VALUE", "ON");
  }

  IupRefresh(ih);  /* refresh the dialog layout */
}


/********************************** Callbacks ************************************


int canvas_action_cb(Ihandle* canvas)
{
  int x, y, canvas_width, canvas_height;
  unsigned int ri, gi, bi;
  imImage* image;
  cdCanvas* cd_canvas = (cdCanvas*)IupGetAttribute(canvas, "cdCanvas");
  Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
  const char* background = IupConfigGetVariableStrDef(config, "Canvas", "Background

  IupGetIntInt(canvas, "DRAWSIZE", &canvas_width, &canvas_height);

  cdCanvasActivate(cd_canvas);

  /* draw the background */
  sscanf(background, "%u␣%u␣%u", &ri, &gi, &bi);
  cdCanvasBackground(cd_canvas, cdEncodeColor((unsigned char)ri, (unsigned char)gi,
  cdCanvasClear(cd_canvas);

  /* draw the image at the center of the canvas */
```

```
  image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  if (image)
  {
    x = (canvas_width - image->width) / 2;
    y = (canvas_height - image->height) / 2;

    imcdCanvasPutImage(cd_canvas, image, x, y, image->width, image->height, 0, 0, 0
  }

  cdCanvasFlush(cd_canvas);
  return IUP_DEFAULT;
}

int canvas_map_cb(Ihandle* canvas)
{
  cdCanvas* cd_canvas = cdCreateCanvas(CD_IUPDBUFFER, canvas);
  IupSetAttribute(canvas, "cdCanvas", (char*)cd_canvas);
  return IUP_DEFAULT;
}

int canvas_unmap_cb(Ihandle* canvas)
{
  cdCanvas* cd_canvas = (cdCanvas*)IupGetAttribute(canvas, "cdCanvas");
  cdKillCanvas(cd_canvas);
  return IUP_DEFAULT;
}

int dropfiles_cb(Ihandle* ih, const char* filename)
{
  if (save_check(ih))
    open_file(ih, filename);

  return IUP_DEFAULT;
}

int file_menu_open_cb(Ihandle* ih)
{
  Ihandle* item_revert = IupGetDialogChild(ih, "ITEM_REVERT");
  Ihandle* item_save = IupGetDialogChild(ih, "ITEM_SAVE");
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  char* filename = IupGetAttribute(canvas, "FILENAME");
  int dirty = IupGetInt(canvas, "DIRTY");
```

```
  if (dirty)
    IupSetAttribute(item_save, "ACTIVE", "YES");
  else
    IupSetAttribute(item_save, "ACTIVE", "NO");

  if (dirty && filename)
    IupSetAttribute(item_revert, "ACTIVE", "YES");
  else
    IupSetAttribute(item_revert, "ACTIVE", "NO");
  return IUP_DEFAULT;
}

int edit_menu_open_cb(Ihandle* ih)
{
  Ihandle *clipboard = IupClipboard();

  Ihandle *item_paste = IupGetDialogChild(ih, "ITEM_PASTE");

  if (!IupGetInt(clipboard, "IMAGEAVAILABLE"))
    IupSetAttribute(item_paste, "ACTIVE", "NO");
  else
    IupSetAttribute(item_paste, "ACTIVE", "YES");

  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int config_recent_cb(Ihandle* ih)
{
  if (save_check(ih))
  {
    char* filename = IupGetAttribute(ih, "TITLE");
    open_file(ih, filename);
  }
  return IUP_DEFAULT;
}

int item_new_action_cb(Ihandle* item_new)
{
  if (save_check(item_new))
  {
```

```
    Ihandle* canvas = IupGetDialogChild(item_new, "CANVAS");
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
    int width = IupConfigGetVariableIntDef(config, "NewImage", "Width", 640);
    int height = IupConfigGetVariableIntDef(config, "NewImage", "Height", 480);

    if (IupGetParam("New Image", NULL, NULL, "Width: %i[1,]\nHeight: %i[1,]\n", &wi
    {
      imImage* image = imImageCreate(width, height, IM_RGB, IM_BYTE);

      IupConfigSetVariableInt(config, "NewImage", "Width", width);
      IupConfigSetVariableInt(config, "NewImage", "Height", height);

      new_file(item_new, image);
    }
  }

  return IUP_DEFAULT;
}

int select_file(Ihandle* parent_dlg, int is_open)
{
  Ihandle* config = (Ihandle*)IupGetAttribute(parent_dlg, "CONFIG");
  Ihandle* canvas = IupGetDialogChild(parent_dlg, "CANVAS");
  const char* dir = IupConfigGetVariableStr(config, "MainWindow", "LastDirectory");

  Ihandle* filedlg = IupFileDlg();
  if (is_open)
    IupSetAttribute(filedlg, "DIALOGTYPE", "OPEN");
  else
  {
    IupSetAttribute(filedlg, "DIALOGTYPE", "SAVE");
    IupSetStrAttribute(filedlg, "FILE", IupGetAttribute(canvas, "FILENAME"));
  }
  IupSetAttribute(filedlg, "EXTFILTER", "Image Files|*.bmp;*.jpg;*.png;*.tif;*.tga|
  IupSetStrAttribute(filedlg, "DIRECTORY", dir);
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", parent_dlg);

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    if (is_open)
```

```
      open_file(parent_dlg, filename);
    else
      saveas_file(canvas, filename);

    dir = IupGetAttribute(filedlg, "DIRECTORY");
    IupConfigSetVariableStr(config, "MainWindow", "LastDirectory", dir);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

int item_open_action_cb(Ihandle* item_open)
{
  if (!save_check(item_open))
    return IUP_DEFAULT;

  return select_file(IupGetDialog(item_open), 1);
}

int item_saveas_action_cb(Ihandle* item_saveas)
{
  return select_file(IupGetDialog(item_saveas), 0);
}

int item_save_action_cb(Ihandle* item_save)
{
  Ihandle* canvas = IupGetDialogChild(item_save, "CANVAS");
  char* filename = IupGetAttribute(canvas, "FILENAME");
  if (!filename)
    item_saveas_action_cb(item_save);
  else
  {
    /* test again because in can be called using the hot key */
    int dirty = IupGetInt(canvas, "DIRTY");
    if (dirty)
      save_file(canvas);
  }
  return IUP_DEFAULT;
}

int item_revert_action_cb(Ihandle* item_revert)
```

```
{
  Ihandle* canvas = IupGetDialogChild(item_revert, "CANVAS");
  char* filename = IupGetAttribute(canvas, "FILENAME");
  open_file(item_revert, filename);
  return IUP_DEFAULT;
}

int item_pagesetup_action_cb(Ihandle* item_pagesetup)
{
  Ihandle* canvas = IupGetDialogChild(item_pagesetup, "CANVAS");
  Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
  int margin_width = IupConfigGetVariableIntDef(config, "Print", "MarginWidth", 20)
  int margin_height = IupConfigGetVariableIntDef(config, "Print", "MarginHeight", 2

  if (IupGetParam("Page␣Setup", NULL, NULL, "Margin␣Width␣(mm):␣%i[1,]\nMargin␣Heig
  {
    IupConfigSetVariableInt(config, "Print", "MarginWidth", margin_width);
    IupConfigSetVariableInt(config, "Print", "MarginHeight", margin_height);
  }

  return IUP_DEFAULT;
}

void view_fit_rect(int canvas_width, int canvas_height, int image_width, int image_
{
  *view_width = canvas_width;
  *view_height = (canvas_width * image_height) / image_width;

  if (*view_height > canvas_height)
  {
    *view_height = canvas_height;
    *view_width = (canvas_height * image_width) / image_height;
  }
}

int item_print_action_cb(Ihandle* item_print)
{
  Ihandle* canvas = IupGetDialogChild(item_print, "CANVAS");
  unsigned int ri, gi, bi;
  imImage* image;
  char* title = IupGetAttribute(IupGetDialog(item_print), "TITLE");
  Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
```

```
  const char* background = IupConfigGetVariableStrDef(config, "Canvas", "Background

  cdCanvas* cd_canvas = cdCreateCanvasf(CD_PRINTER, "%s -d", title);
  if (!cd_canvas)
    return IUP_DEFAULT;

  /* draw the background */
  sscanf(background, "%u %u %u", &ri, &gi, &bi);
  cdCanvasBackground(cd_canvas, cdEncodeColor((unsigned char)ri, (unsigned char)gi,
  cdCanvasClear(cd_canvas);

  /* draw the image at the center of the canvas */
  image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  if (image)
  {
    int x, y, canvas_width, canvas_height, view_width, view_height;
    double canvas_width_mm, canvas_height_mm;
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
    int margin_width = IupConfigGetVariableIntDef(config, "Print", "MarginWidth", 2
    int margin_height = IupConfigGetVariableIntDef(config, "Print", "MarginHeight",

    cdCanvasGetSize(cd_canvas, &canvas_width, &canvas_height, &canvas_width_mm, &ca

    /* convert to pixels */
    margin_width = (int)((margin_width * canvas_width) / canvas_width_mm);
    margin_height = (int)((margin_height * canvas_height) / canvas_height_mm);

    view_fit_rect(canvas_width - 2 * margin_width, canvas_height - 2 * margin_heigh
                  image->width, image->height,
                  &view_width, &view_height);

    x = (canvas_width - view_width) / 2;
    y = (canvas_height - view_height) / 2;

    imcdCanvasPutImage(cd_canvas, image, x, y, view_width, view_height, 0, 0, 0, 0)
  }

  cdKillCanvas(cd_canvas);
  return IUP_DEFAULT;
}

int item_exit_action_cb(Ihandle* item_exit)
```

```
{
  Ihandle* dlg = IupGetDialog(item_exit);
  Ihandle* config = (Ihandle*)IupGetAttribute(dlg, "CONFIG");
  Ihandle* canvas = IupGetDialogChild(dlg, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");

  if (!save_check(item_exit))
    return IUP_IGNORE;  /* to abort the CLOSE_CB callback */

  if (image)
    imImageDestroy(image);

  IupConfigDialogClosed(config, dlg, "MainWindow");
  IupConfigSave(config);
  IupDestroy(config);
  return IUP_CLOSE;
}

int item_copy_action_cb(Ihandle* item_copy)
{
  Ihandle* canvas = IupGetDialogChild(item_copy, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(clipboard, "NATIVEIMAGE", (char*)IupGetImageNativeHandle(image));
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_paste_action_cb(Ihandle* item_paste)
{
  if (save_check(item_paste))
  {
    Ihandle* canvas = IupGetDialogChild(item_paste, "CANVAS");
    imImage* old_image = (imImage*)IupGetAttribute(canvas, "IMAGE");

    Ihandle *clipboard = IupClipboard();
    imImage* image = IupGetNativeHandleImage(IupGetAttribute(clipboard, "NATIVEIMAG
    IupDestroy(clipboard);

    if (!image)
    {
      show_error("Invalid␣Clipboard␣Data", 1);
```

```
      return IUP_DEFAULT;
    }

    /* we are going to support only RGB images with no alpha */
    imImageRemoveAlpha(image);
    if (image->color_space != IM_RGB)
    {
      imImage* new_image = imImageCreateBased(image, -1, -1, IM_RGB, -1);
      imConvertColorSpace(image, new_image);
      imImageDestroy(image);

      image = new_image;
    }

    imImageSetAttribString(image, "FileFormat", "JPEG");

    IupSetAttribute(canvas, "DIRTY", "Yes");
    IupSetAttribute(canvas, "IMAGE", (char*)image);
    IupSetAttribute(canvas, "FILENAME", NULL);
    IupSetAttribute(IupGetDialog(canvas), "TITLE", "Untitled - Simple Paint");

    IupUpdate(canvas);

    if (old_image)
      imImageDestroy(old_image);
  }
  return IUP_DEFAULT;
}

int item_background_action_cb(Ihandle* item_background)
{
  Ihandle* canvas = IupGetDialogChild(item_background, "CANVAS");
  Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
  Ihandle* colordlg = IupColorDlg();
  const char* background = IupConfigGetVariableStrDef(config, "Canvas", "Background
  IupSetStrAttribute(colordlg, "VALUE", background);
  IupSetAttributeHandle(colordlg, "PARENTDIALOG", IupGetDialog(item_background));

  IupPopup(colordlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(colordlg, "STATUS") == 1)
  {
```

```
    background = IupGetAttribute(colordlg, "VALUE");
    IupConfigSetVariableStr(config, "Canvas", "Background", background);

    IupUpdate(canvas);
  }

  IupDestroy(colordlg);
  return IUP_DEFAULT;
}

int item_toolbar_action_cb(Ihandle* item_toolbar)
{
  Ihandle* canvas = IupGetDialogChild(item_toolbar, "CANVAS");
  Ihandle* toolbar = IupGetChild(IupGetParent(canvas), 0);
  Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");

  toggle_bar_visibility(item_toolbar, toolbar);

  IupConfigSetVariableStr(config, "MainWindow", "Toolbar", IupGetAttribute(item_too
  return IUP_DEFAULT;
}

int item_statusbar_action_cb(Ihandle* item_statusbar)
{
  Ihandle* canvas = IupGetDialogChild(item_statusbar, "CANVAS");
  Ihandle* statusbar = IupGetBrother(canvas);
  Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");

  toggle_bar_visibility(item_statusbar, statusbar);

  IupConfigSetVariableStr(config, "MainWindow", "Statusbar", IupGetAttribute(item_s
  return IUP_DEFAULT;
}

int item_help_action_cb(void)
{
  IupHelp("http://www.tecgraf.puc-rio.br/iup");
  return IUP_DEFAULT;
}

int item_about_action_cb(void)
{
```

```
  IupMessage("About", "   Simple Paint\n\nAutors:\n   Gustavo Lyrio\n   Antonio Scu
  return IUP_DEFAULT;
}

/********************************* Main ****************************************/

Ihandle* create_main_dialog(Ihandle *config)
{
  Ihandle *dlg, *vbox, *canvas, *menu;
  Ihandle *sub_menu_file, *file_menu, *item_exit, *item_new, *item_open, *item_save
  Ihandle *sub_menu_edit, *edit_menu, *item_copy, *item_paste, *item_print, *item_p
  Ihandle *btn_copy, *btn_paste, *btn_new, *btn_open, *btn_save;
  Ihandle *sub_menu_help, *help_menu, *item_help, *item_about;
  Ihandle *sub_menu_view, *view_menu, *item_toolbar, *item_statusbar;
  Ihandle *statusbar, *toolbar, *recent_menu, *item_background;

  canvas = IupCanvas(NULL);
  IupSetAttribute(canvas, "NAME", "CANVAS");
  IupSetAttribute(canvas, "DIRTY", "NO");
  IupSetCallback(canvas, "ACTION", (Icallback)canvas_action_cb);
  IupSetCallback(canvas, "DROPFILES_CB", (Icallback)dropfiles_cb);
  IupSetCallback(canvas, "MAP_CB", (Icallback)canvas_map_cb);
  IupSetCallback(canvas, "UNMAP_CB", (Icallback)canvas_unmap_cb);

  statusbar = IupLabel("(0, 0) = [0   0   0]");
  IupSetAttribute(statusbar, "NAME", "STATUSBAR");
  IupSetAttribute(statusbar, "EXPAND", "HORIZONTAL");
  IupSetAttribute(statusbar, "PADDING", "10x5");

  item_new = IupItem("&New\tCtrl+N", NULL);
  IupSetAttribute(item_new, "IMAGE", "IUP_FileNew");
  IupSetCallback(item_new, "ACTION", (Icallback)item_new_action_cb);
  btn_new = IupButton(NULL, NULL);
  IupSetAttribute(btn_new, "IMAGE", "IUP_FileNew");
  IupSetAttribute(btn_new, "FLAT", "Yes");
  IupSetCallback(btn_new, "ACTION", (Icallback)item_new_action_cb);
  IupSetAttribute(btn_new, "TIP", "New (Ctrl+N)");
  IupSetAttribute(btn_new, "CANFOCUS", "No");

  item_open = IupItem("&Open...\tCtrl+O", NULL);
```

```
IupSetAttribute(item_open, "IMAGE", "IUP_FileOpen");
IupSetCallback(item_open, "ACTION", (Icallback)item_open_action_cb);
btn_open = IupButton(NULL, NULL);
IupSetAttribute(btn_open, "IMAGE", "IUP_FileOpen");
IupSetAttribute(btn_open, "FLAT", "Yes");
IupSetCallback(btn_open, "ACTION", (Icallback)item_open_action_cb);
IupSetAttribute(btn_open, "TIP", "Open (Ctrl+O)");
IupSetAttribute(btn_open, "CANFOCUS", "No");

item_save = IupItem("&Save\tCtrl+S", NULL);
IupSetAttribute(item_save, "NAME", "ITEM_SAVE");
IupSetAttribute(item_save, "IMAGE", "IUP_FileSave");
IupSetCallback(item_save, "ACTION", (Icallback)item_save_action_cb);
btn_save = IupButton(NULL, NULL);
IupSetAttribute(btn_save, "IMAGE", "IUP_FileSave");
IupSetAttribute(btn_save, "FLAT", "Yes");
IupSetCallback(btn_save, "ACTION", (Icallback)item_save_action_cb);
IupSetAttribute(btn_save, "TIP", "Save (Ctrl+S)");
IupSetAttribute(btn_save, "CANFOCUS", "No");

item_saveas = IupItem("Save &As...", NULL);
IupSetAttribute(item_saveas, "NAME", "ITEM_SAVEAS");
IupSetCallback(item_saveas, "ACTION", (Icallback)item_saveas_action_cb);

item_revert = IupItem("&Revert", NULL);
IupSetAttribute(item_revert, "NAME", "ITEM_REVERT");
IupSetCallback(item_revert, "ACTION", (Icallback)item_revert_action_cb);

item_pagesetup = IupItem("Page Set&up...", NULL);
IupSetCallback(item_pagesetup, "ACTION", (Icallback)item_pagesetup_action_cb);

item_print = IupItem("&Print...\tCtrl+P", NULL);
IupSetCallback(item_print, "ACTION", (Icallback)item_print_action_cb);

item_exit = IupItem("E&xit", NULL);
IupSetCallback(item_exit, "ACTION", (Icallback)item_exit_action_cb);

item_copy = IupItem("&Copy\tCtrl+C", NULL);
IupSetAttribute(item_copy, "NAME", "ITEM_COPY");
IupSetAttribute(item_copy, "IMAGE", "IUP_EditCopy");
IupSetCallback(item_copy, "ACTION", (Icallback)item_copy_action_cb);
btn_copy = IupButton(NULL, NULL);
```

```
  IupSetAttribute(btn_copy, "IMAGE", "IUP_EditCopy");
  IupSetAttribute(btn_copy, "FLAT", "Yes");
  IupSetCallback(btn_copy, "ACTION", (Icallback)item_copy_action_cb);
  IupSetAttribute(btn_copy, "TIP", "Copy (Ctrl+C)");
  IupSetAttribute(btn_copy, "CANFOCUS", "No");

  item_paste = IupItem("&Paste\tCtrl+V", NULL);
  IupSetAttribute(item_paste, "NAME", "ITEM_PASTE");
  IupSetAttribute(item_paste, "IMAGE", "IUP_EditPaste");
  IupSetCallback(item_paste, "ACTION", (Icallback)item_paste_action_cb);
  btn_paste = IupButton(NULL, NULL);
  IupSetAttribute(btn_paste, "IMAGE", "IUP_EditPaste");
  IupSetAttribute(btn_paste, "FLAT", "Yes");
  IupSetCallback(btn_paste, "ACTION", (Icallback)item_paste_action_cb);
  IupSetAttribute(btn_paste, "TIP", "Paste (Ctrl+V)");
  IupSetAttribute(btn_paste, "CANFOCUS", "No");

  item_background = IupItem("&Background...", NULL);
  IupSetCallback(item_background, "ACTION", (Icallback)item_background_action_cb);

  item_toolbar = IupItem("&Toobar", NULL);
  IupSetCallback(item_toolbar, "ACTION", (Icallback)item_toolbar_action_cb);
  IupSetAttribute(item_toolbar, "VALUE", "ON");

  item_statusbar = IupItem("&Statusbar", NULL);
  IupSetCallback(item_statusbar, "ACTION", (Icallback)item_statusbar_action_cb);
  IupSetAttribute(item_statusbar, "VALUE", "ON");

  item_help = IupItem("&Help...", NULL);
  IupSetCallback(item_help, "ACTION", (Icallback)item_help_action_cb);

  item_about = IupItem("&About...", NULL);
  IupSetCallback(item_about, "ACTION", (Icallback)item_about_action_cb);

  recent_menu = IupMenu(NULL);

  file_menu = IupMenu(
    item_new,
    item_open,
    item_save,
    item_saveas,
    item_revert,
```

```
    IupSeparator(),
    item_pagesetup,
    item_print,
    IupSeparator(),
    IupSubmenu("Recent &Files", recent_menu),
    IupSeparator(),
    item_exit,
    NULL);
  edit_menu = IupMenu(
    item_copy,
    item_paste,
    NULL);
  view_menu = IupMenu(
    item_background,
    IupSeparator(),
    item_toolbar,
    item_statusbar,
    NULL);
  help_menu = IupMenu(
    item_help,
    item_about,
    NULL);

  IupSetCallback(file_menu, "OPEN_CB", (Icallback)file_menu_open_cb);
  IupSetCallback(edit_menu, "OPEN_CB", (Icallback)edit_menu_open_cb);

  sub_menu_file = IupSubmenu("&File", file_menu);
  sub_menu_edit = IupSubmenu("&Edit", edit_menu);
  sub_menu_view = IupSubmenu("&View", view_menu);
  sub_menu_help = IupSubmenu("&Help", help_menu);

  menu = IupMenu(
    sub_menu_file,
    sub_menu_edit,
    sub_menu_view,
    sub_menu_help,
    NULL);

  toolbar = IupHbox(
    btn_new,
    btn_open,
    btn_save,
```

```
    IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
    btn_copy,
    btn_paste,
    NULL);
  IupSetAttribute(toolbar, "MARGIN", "5x5");
  IupSetAttribute(toolbar, "GAP", "2");

  vbox = IupVbox(
    toolbar,
    canvas,
    statusbar,
    NULL);

  dlg = IupDialog(vbox);
  IupSetAttributeHandle(dlg, "MENU", menu);
  IupSetAttribute(dlg, "SIZE", "HALFxHALF");
  IupSetCallback(dlg, "CLOSE_CB", (Icallback)item_exit_action_cb);
  IupSetCallback(dlg, "DROPFILES_CB", (Icallback)dropfiles_cb);

  IupSetCallback(dlg, "K_cN", (Icallback)item_new_action_cb);
  IupSetCallback(dlg, "K_cO", (Icallback)item_open_action_cb);
  IupSetCallback(dlg, "K_cS", (Icallback)item_save_action_cb);
  IupSetCallback(dlg, "K_cV", (Icallback)item_paste_action_cb);
  IupSetCallback(dlg, "K_cC", (Icallback)item_copy_action_cb);
  IupSetCallback(dlg, "K_cP", (Icallback)item_print_action_cb);

  /* parent for pre-defined dialogs in closed functions (IupMessage and IupAlarm) *
  IupSetAttributeHandle(NULL, "PARENTDIALOG", dlg);

  /* Initialize variables from the configuration file */

  IupConfigRecentInit(config, recent_menu, config_recent_cb, 10);

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Toolbar", 1))
  {
    IupSetAttribute(item_toolbar, "VALUE", "OFF");

    IupSetAttribute(toolbar, "FLOATING", "YES");
    IupSetAttribute(toolbar, "VISIBLE", "NO");
  }

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Statusbar", 1))
```

```
  {
    IupSetAttribute(item_statusbar, "VALUE", "OFF");

    IupSetAttribute(statusbar, "FLOATING", "YES");
    IupSetAttribute(statusbar, "VISIBLE", "NO");
  }

  IupSetAttribute(dlg, "CONFIG", (char*)config);

  return dlg;
}

int main(int argc, char **argv)
{
  Ihandle *dlg;
  Ihandle *config;

  IupOpen(&argc, &argv);
  IupImageLibOpen();

  config = IupConfig();
  IupSetAttribute(config, "APP_NAME", "simple_paint");
  IupConfigLoad(config);

  dlg = create_main_dialog(config);

  /* show the dialog at the last position, with the last size */
  IupConfigDialogShow(config, dlg, "MainWindow");

  /* open a file from the command line (allow file association in Windows) */
  if (argc > 1 && argv[1])
  {
    const char* filename = argv[1];
    open_file(dlg, filename);
  }

  /* initialize the current file, if not already loaded */
  check_new_file(dlg);

  IupMainLoop();

  IupClose();
```

```
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("imlua")
require("iuplua")
require("iupluaimglib")
require("iupluaim")
require("cdlua")
require("cdluaim")
require("iupluacd")


--********************************** Utilities ******************************************

function str_splitfilename(filename)
  return string.match(filename, "(.-)([^\\/]-%.?([^%.\\/]*))$")
end

function str_fileext(filename)
  local path, title, ext = str_splitfilename(filename)
  return ext
end

function str_filetitle(filename)
  local path, title, ext = str_splitfilename(filename)
  return title
end

function show_error(message, is_error)
  local dlg = iup.messagedlg{
    parentdialog = iup.GetGlobal("PARENTDIALOG"),
    buttons = "OK",
    value = message,
  }
  if (is_error) then
    dlg.dialogtype = "ERROR"
    dlg.title = "Error"
  else
    dlg.dialogtype = "WARNING"
    dlg.title = "Warning"
  end
```

```
  dlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)
  dlg:destroy()
end

function read_file(filename)
  local image, err = im.FileImageLoadBitmap(filename, 0)
  if (err) then
    show_error(im.ErrorStr(err), true)
  else
    -- we are going to support only RGB images with no alpha
    image:RemoveAlpha()
    if (image:ColorSpace() ~= im.RGB) then
      local new_image = im.ImageCreateBased(image, nil, nil, im.RGB, nil)

      im.ConvertColorSpace(image, new_image)
      image:Destroy()

      image = new_image
    end
  end
  return image
end

function write_file(filename, image)
  local format = image:GetAttribString("FileFormat")
  local err = im.FileImageSave(filename, format, image)
  if (err and err ~= im.ERR_NONE) then
    show_error(im.ErrorStr(err), true)
    return false
  end
  return true
end

function new_file(ih, image)
  local dlg = iup.GetDialog(ih)
  local canvas = dlg.canvas
  local old_image = canvas.image

  dlg.title = "Untitled - Simple Paint"
  canvas.filename = nil
  canvas.dirty = nil
  canvas.image = image
```

```
  iup.Update(canvas)

  if (old_image) then
    old_image:Destroy()
  end
end

function check_new_file(dlg)
  local canvas = dlg.canvas
  local image = canvas.image
  if (not image) then
    local config = canvas.config
    local width = config:GetVariableDef("NewImage", "Width", 640)
    local height = config:GetVariableDef("NewImage", "Height", 480)

    local image = im.ImageCreate(width, height, im.RGB, im.BYTE)

    new_file(dlg, image)
  end
end

function open_file(ih, filename)
  local image = read_file(filename)
  if (image) then
    local dlg = iup.GetDialog(ih)
    local canvas = dlg.canvas
    local config = canvas.config
    local old_image = canvas.image

    dlg.title = str_filetitle(filename).." - Simple Paint"
    canvas.filename = filename
    canvas.dirty = nil
    canvas.image = image

    iup.Update(canvas)

    if (old_image) then
      old_image:Destroy()
    end

    config:RecentUpdate(filename)
```

```
  end
end

function save_file(canvas)
  if (write_file(canvas.filename, canvas.image)) then
    canvas.dirty = nil
  end
end

function set_file_format(image, filename)
  local ext = str_fileext(filename)
  ext:lower()
  local format = "JPEG"
  if (ext == "jpg" or ext == "jpeg") then
    format = "JPEG"
  elseif (ext == "bmp") then
    format = "BMP"
  elseif (ext == "png") then
    format = "PNG"
  elseif (ext == "tga") then
    format = "TGA"
  elseif (ext == "tif" or ext == "tiff") then
    format = "TIFF"
  end
  image:SetAttribString("FileFormat", format)
end

function saveas_file(canvas, filename)
  local image = canvas.image

  set_file_format(image, filename)

  if (write_file(filename, image)) then
    local dlg = iup.GetDialog(canvas)
    local config = canvas.config

    dlg.title = str_filetitle(filename).." - Simple Paint"
    canvas.filename = filename
    canvas.dirty = nil

    config:RecentUpdate(filename)
  end
```

```
end

function save_check(ih)
  local dlg = iup.GetDialog(ih)
  local canvas = dlg.canvas

  if (canvas.dirty) then
    local resp = iup.Alarm("Warning", "File not saved! Save it now?", "Yes", "No",
    if resp == 1 then -- save the changes and continue
      save_file(canvas)
    elseif resp == 3 then  -- cancel
      return false
    else  -- ignore the changes and continue
    end
  end
  return true
end

function toggle_bar_visibility(item, bar)
  if (item.value == "ON") then
    bar.floating = "YES"
    bar.visible = "NO"
    item.value = "OFF"
  else
    bar.floating = "NO"
    bar.visible = "YES"
    item.value = "ON"
  end
  iup.Refresh(bar)  -- refresh the dialog layout
end


--******************************** Main (Part 1/2) ****************************

-- create all the elements that will have callbacks in Lua prior to callbacks defin

config = iup.config{}
config.app_name = "simple_paint"
config:Load()

statusbar = iup.label{title = "(0, 0) = [0   0   0]", expand = "HORIZONTAL", paddin
```

```
canvas = iup.canvas{
  config = config,
  dirty = nil,
}

item_new = iup.item{title = "&New...\tCtrl+N", image = "IUP_FileNew"}
item_open = iup.item{title = "&Open...\tCtrl+O", image = "IUP_FileOpen"}
item_save = iup.item{title="&Save\tCtrl+S"}
item_saveas = iup.item{title="Save &As...", image = "IUP_FileSave"}
item_revert = iup.item{title="&Revert"}
item_pagesetup = iup.item{title="Page Set&up..."}
item_print = iup.item{title="&Print\tCtrl+P"}
item_exit = iup.item{title="E&xit"}
item_copy = iup.item{title="&Copy\tCtrl+C", image = "IUP_EditCopy"}
item_paste = iup.item{title="&Paste\tCtrl+V", image = "IUP_EditPaste"}
item_background = iup.item{title="&Background..."}
item_toolbar = iup.item{title="&Toobar", value="ON"}
item_statusbar = iup.item{title="&Statusbar", value="ON"}
item_help = iup.item{title="&Help..."}
item_about = iup.item{title="&About..."}

recent_menu = iup.menu{}

file_menu = iup.menu{
  item_new,
  item_open,
  item_save,
  item_saveas,
  item_revert,
  iup.separator{},
  item_pagesetup,
  item_print,
  iup.separator{},
  iup.submenu{title="Recent &Files", recent_menu},
  iup.separator{},
  item_exit
  }

edit_menu = iup.menu{
  item_copy,
  item_paste,
  }
```

```
view_menu = iup.menu{
  item_background,
  iup.separator{},
  item_toolbar,
  item_statusbar,
  }
help_menu = iup.menu{item_help, item_about}

sub_menu_file = iup.submenu{file_menu, title = "&File"}
sub_menu_edit = iup.submenu{edit_menu, title = "&Edit"}
sub_menu_view = iup.submenu{title = "&View", view_menu}
sub_menu_help = iup.submenu{help_menu, title = "&Help"}

menu = iup.menu{
  sub_menu_file,
  sub_menu_edit,
  sub_menu_view,
  sub_menu_help,
  }


--********************************** Callbacks ************************************


function canvas:action()
  local image = canvas.image
  local canvas_width, canvas_height = string.match(canvas.drawsize,"(%d*)x(%d*)")
  local cd_canvas = canvas.cdCanvas

  cd_canvas:Activate()

  -- draw the background
  local background = config:GetVariableDef("Canvas", "Background", "255 255 255")
  local r, g, b = string.match(background, "(%d*) (%d*) (%d*)")
  cd_canvas:Background(cd.EncodeColor(r, g, b))
  cd_canvas:Clear()

  -- draw the image at the center of the canvas
  if (image) then
    local x = math.floor((canvas_width - image:Width()) / 2)
    local y = math.floor((canvas_height - image:Height()) / 2)
```

```
    image:cdCanvasPutImageRect(cd_canvas, x, y, image:Width(), image:Height(), 0, 0
  end

  cd_canvas:Flush()
end

function canvas:map_cb()
  cd_canvas = cd.CreateCanvas(cd.IUPDBUFFER, canvas)
  canvas.cdCanvas = cd_canvas
end

function canvas:unmap_cb()
  local cd_canvas = canvas.cdCanvas
  cd_canvas:Kill()
end

function canvas:dropfiles_cb(filename)
  if (save_check(self)) then
    open_file(self, filename)
  end
end

function file_menu:open_cb()
  if (canvas.dirty) then
    item_save.active = "YES"
  else
    item_save.active = "NO"
  end
  if (canvas.dirty and canvas.filename) then
    item_revert.active = "YES"
  else
    item_revert.active = "NO"
  end
end

function edit_menu:open_cb()
  local clipboard = iup.clipboard{}
  if (clipboard.imageavailable == "NO") then
    item_paste.active = "NO"
  else
    item_paste.active = "YES"
  end
```

```
  clipboard:destroy()
end

function config:recent_cb()
  if (save_check(self)) then
    local filename = self.title
    open_file(self, filename)
  end
end

function item_new:action()
  if save_check(self) then
    local width = config:GetVariableDef("NewImage", "Width", 640)
    local height = config:GetVariableDef("NewImage", "Height", 480)

    local ret, new_width, new_height = iup.GetParam("New Image", nil, "Width: %i[1,
    if (ret) then
      local new_image = im.ImageCreate(new_width, new_height, im.RGB, im.BYTE)

      config:SetVariable("NewImage", "Width", new_width)
      config:SetVariable("NewImage", "Height", new_height)

      new_file(item_new, new_image)
    end
  end
end

function select_file(parent_dlg, is_open)
  local filedlg = iup.filedlg{
    extfilter="Image Files|*.bmp;*.jpg;*.png;*.tif;*.tga|All Files|*.*|",
    parentdialog = parent_dlg,
    directory = config:GetVariable("MainWindow", "LastDirectory"),
    }

  if (is_open) then
    filedlg.dialogtype = "OPEN"
  else
    filedlg.dialogtype = "SAVE"
    filedlg.file = canvas.filename
  end

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)
```

```
  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    if (is_open) then
      open_file(parent_dlg, filename)
    else
      saveas_file(canvas, filename)
    end

    config:SetVariable("MainWindow", "LastDirectory", filedlg.directory)
  end

  filedlg:destroy()
end

function item_open:action()
  if not save_check(self) then
    return
  end

  select_file(dlg, true)
end

function item_saveas:action()
  select_file(dlg, false)
end

function item_save:action()
  if (not canvas.filename) then
    item_saveas:action()
  else
    -- test again because in can be called using the hot key
    if (canvas.dirty) then
      save_file(canvas)
    end
  end
end

function item_revert:action()
  open_file(self, canvas.filename)
end
```

```
function item_pagesetup:action()
  local width = config:GetVariableDef("Print", "MarginWidth", 20)
  local height = config:GetVariableDef("Print", "MarginHeight", 20)

  local ret, new_width, new_height = iup.GetParam("Page Setup", nil, "nMargin Width
  if (ret) then
    config:SetVariable("Print", "MarginWidth", new_width)
    config:SetVariable("Print", "MarginHeight", new_height)
  end
end

function view_fit_rect(canvas_width, canvas_height, image_width, image_height)
  local view_width = canvas_width
  local view_height = (canvas_width * image_height) / image_width

  if (view_height > canvas_height) then
    view_height = canvas_height
    view_width = (canvas_height * image_width) / image_height
  end

  return view_width, view_height
end

function item_print:action()
  local title = dlg.title
  local cd_canvas = cd.CreateCanvas(cd.PRINTER, title.." -d")
  if (not cd_canvas) then
    return
  end

  -- draw the background
  local background = config:GetVariableDef("Canvas", "Background", "255 255 255")
  local r, g, b = string.match(background, "(%d*) (%d*) (%d*)")
  cd_canvas:Background(cd.EncodeColor(r, g, b))
  cd_canvas:Clear()

  -- draw the image at the center of the canvas
  local image = canvas.image
  if (image) then
    local margin_width = config:GetVariableDef("Print", "MarginWidth", 20)
    local margin_height = config:GetVariableDef("Print", "MarginHeight", 20)
```

```
    local canvas_width, canvas_height, canvas_width_mm, canvas_height_mm = cd_canva

    -- convert to pixels
    margin_width = (margin_width * canvas_width) / canvas_width_mm
    margin_height = (margin_height * canvas_height) / canvas_height_mm

    local view_width, view_height = view_fit_rect(
      canvas_width - 2 * margin_width, canvas_height - 2 * margin_height,
      image:Width(), image:Height())

    local x = math.floor((canvas_width - view_width) / 2)
    local y = math.floor((canvas_height - view_height) / 2)

    image:cdCanvasPutImageRect(cd_canvas, x, y, view_width, view_height, 0, 0, 0, 0
  end

  cd_canvas:Kill()
end

function item_exit:action()
  local image = canvas.image

  if not save_check(self) then
    return iup.IGNORE  -- to abort the CLOSE_CB callback
  end

  if (image) then
    image:Destroy()
  end

  config:DialogClosed(iup.GetDialog(self), "MainWindow")
  config:Save()
  config:destroy()
  return iup.CLOSE
end

function item_copy:action()
  local clipboard = iup.clipboard{}
  -- must use iup.SetAttribute because it is an userdata
  iup.SetAttribute(clipboard, "NATIVEIMAGE", iup.GetImageNativeHandle(canvas.image)
  clipboard:destroy()
end
```

```
function item_paste:action()
  if save_check(self) then
    local clipboard = iup.clipboard{}
    local old_image = canvas.image

    local image = iup.GetNativeHandleImage(clipboard.nativeimage)

    -- we are going to support only RGB images with no alpha
    image:RemoveAlpha()
    if (image:ColorSpace() ~= im.RGB) then
      local new_image = im.ImageCreateBased(image, nil, nil, im.RGB, nil)

      im.ConvertColorSpace(image, new_image)
      image:Destroy()

      image = new_image
    end

    image:SetAttribString("FileFormat", "JPEG")

    canvas.dirty = "Yes"
    canvas.image = image
    canvas.filename = nil
    dlg.title = "Untitled - Simple Paint"

    iup.Update(canvas)

    if (old_image) then
      old_image:Destroy()
    end

    clipboard:destroy()
  end
end

function item_background:action()
  local colordlg = iup.colordlg{}
  local background = config:GetVariableDef("Canvas", "Background", "255 255 255")
  colordlg.value = background
  colordlg.parentdialog = iup.GetDialog(self)
```

```
  colordlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(colordlg.status) == 1) then
    background = colordlg.value
    config:SetVariable("Canvas", "Background", background)

    iup.Update(canvas)
  end

  colordlg:destroy()
end

function item_toolbar:action()
  toggle_bar_visibility(self, toolbar)
  config:SetVariable("MainWindow", "Toolbar", item_toolbar.value)
end

function item_statusbar:action()
  toggle_bar_visibility(self, statusbar)
  config:SetVariable("MainWindow", "Statusbar", item_statusbar.value)
end

function item_help:action()
  iup.Help("http://www.tecgraf.puc-rio.br/iup")
end

function item_about:action()
  iup.Message("About", "␣␣␣Simple␣Paint\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣Sc
end


--********************************* Main (Part 2/2) ****************************


btn_new = iup.button{image = "IUP_FileNew", flat = "Yes", action = item_new.action,
btn_open = iup.button{image = "IUP_FileOpen", flat = "Yes", action = item_open.actio
btn_save = iup.button{image = "IUP_FileSave", flat = "Yes", action = item_save.actio
btn_copy = iup.button{image =  "IUP_EditCopy", flat = "Yes", action = item_copy.act
btn_paste = iup.button{image = "IUP_EditPaste", flat = "Yes", action = item_paste.a

toolbar = iup.hbox{
  btn_new,
```

```
  btn_open,
  btn_save,
  iup.label{separator="VERTICAL"},
  btn_copy,
  btn_paste,
  margin = "5x5",
  gap = 2,
}

vbox = iup.vbox{
  toolbar,
  canvas,
  statusbar,
}

dlg = iup.dialog{
  vbox,
  title = "Simple Paint",
  size = "HALFxHALF",
  menu = menu,
  close_cb = item_exit.action,
  canvas = canvas,
  dropfiles_cb = canvas.dropfiles_cb,
}

function dlg:k_any(c)
  if (c == iup.K_cN) then
    item_new:action()
  elseif (c == iup.K_cO) then
    item_open:action()
  elseif (c == iup.K_cS) then
    item_save:action()
  elseif (c == iup.K_cV) then
    item_paste:action()
  elseif (c == iup.K_cC) then
    item_copy:action()
  elseif (c == iup.K_cP) then
    item_print:action()
  end
end

-- parent for pre-defined dialogs in closed functions (IupMessage and IupAlarm)
```

```
iup.SetGlobal("PARENTDIALOG", iup.SetHandleName(dlg))

-- Initialize variables from the configuration file

config:RecentInit(recent_menu, 10)

show_statusbar = config:GetVariableDef("MainWindow", "Statusbar", "ON")
if (show_statusbar == "OFF") then
  item_statusbar.value = "OFF"
  statusbar.floating = "YES"
  statusbar.visible = "NO"
end

show_toolbar = config:GetVariableDef("MainWindow", "Toolbar", "ON")
if (show_toolbar == "OFF") then
  item_toolbar.value = "OFF"
  toolbar.floating = "YES"
  toolbar.visible = "NO"
end

-- show the dialog at the last position, with the last size
config:DialogShow(dlg, "MainWindow")

-- open a file from the command line (allow file association in Windows)
if (arg and arg[1]) then
  filename = arg[1]
  open_file(dlg, filename)
end

-- initialize the current file, if not already loaded
check_new_file(dlg)

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

4.4 缩放和滚动条

在本节，我们将为我们的程序增加缩放的功能。我们将会以大于或小于图像的实际的尺寸的大小来绘制图像。我们允许用户通过多个途径来修改缩放系数。

当图像大小大于我们的画布时，我们需要滚动条来移动图像的可见部位。通过设置 *IupCanvas*

的 *SCROLLBAR* 属性为 *Yes*，即可使画布拥有一个滚动条。但是，需要记得在缩放系数改变和我们的窗口大小改变时要重新配置这些滚动条。我们通过画布的 *RESIZE_CB* 回调函数来进行滚动条的配置，我们编写了 *scrollbar_update* 函数来计算滚动条的参数。

我们在状态栏上添加了一些控件来使用户能够修改缩放系数：一个 *IupVal* 控件，供用户选择一个指定范围内的值，三个按钮，分别代表缩小，放大，和原始大小。按钮的作用和 *View* 菜单下的对应菜单项目是一样的。我们也为这些按钮绑定了热键：*Ctrl+* 放大，*Ctrl-*缩小，*Ctrl0* 原始大小。最后，我们给画布添加了一个 *WHEEL_CB* 回调，我们通过它响应鼠标滚轮来修改缩放系数。我们可以在 *create_main_dialog* 函数中找到这些新添加的控件。这次，我们使用了更加简洁的方式来创建这些控件。结构上类似于在 Lua 形式下创建这些控件。缩放系数的变化是线性的，但它的效果却是 2 的指数，控件可以修改的范围是 −6 到 6，我们通过 *pow(2,zoom_index)* 来计算真正的缩放系数。−6 到 6 这个范围产生的缩放范围是 16400

scrollbar_update 函数对滚动条进行了配置。可以参考 **IUP 手册**中关于 *SCROLLBAR* 属性的内容。

用户修改缩放系数之后，我们就需要绘制图像。在画布的绘图的回调函数中，我们计算了图像应该绘制在画布上的大小和位置。

我们调用 *cdCanvasRect* 函数来给图像添加了一个边框。这样做的原因是添加缩放和滚动条之后，如果图像的内容和我们的程序背景差不多的时候，我们很难判断图像的边界，添加一个边框后就很容易分辨了。

实施

在 Lua 形式下，数学库已经被自动载入，我们不需要在包含它。其余部分和 C 语言只有语言形式的不同，所以我们就不详细描述了。

4.5 画布和工具栏

4.6 图像处理和最后的讨论

在最后，我们为程序添加一些 **IM** 库自带的一些图像处理功能。我们在主菜单里添加一个叫做 *Image* 的菜单项目，然后在它的下面添加 *Resize*，*Mirror*，*Flip*，*Rotate*，*Negative*,*Brightness*，*Contrast* 菜单项目。我们和之前一样使用 *IupGetParam* 对话框来完成这些功能的参数获取，我们通过 *PARAM_CB* 回调来动态地更新参数调整后的效果到图像上。为了实现这种所见所得的功能，我们使用一个临时的图像来替代当前的图像来进行图形处理，避免对原来的图像造成影响。这个功能的效果非常棒。**IM** 库还提供了其它一些图像处理功能，使用它们需要连接 *im_process* 库。我们也可以把填充替换为 *imProcess* 的版本。

我们还可以使用 **cdim** 库，它允许我们替换 *CD_IMAGERGB* 驱动为 *CD_IMIMAGE* 驱动，*imcdCanvasPutImage* 宏替换为 *cdCanvasPutImImage* 函数。这使我们的代码显得更加优雅。

本章已经快要结束了。我们从短短 800 行的代码开始，到现在的 2500 行。我们使用 **IUP** 和 **CD** 库实现了一个功能全面的画图程序。它包含读取和保存图像文件，绘制和打印图像，缩放和滚动显示的功能，通过它我们演示了如何和画布进行交互，以及在画布上绘图的多种方法。

这个画图程序还有许多可以增强的地方，比如使用 **IM** 库的视频捕捉功能来从相机获取图像，使用栈来实现撤销和重做功能，使用 Alpha 通道来进行透明处理，实现区域选择等等。

工具箱也是一个重要的界面元素。我们不仅可以隐藏它，还可以使用 *IupDetachBox* 来把它插入到主窗口中。工具箱可以有三种状态：隐藏，作为一个独立的对话框，像工具栏一样依附在一个窗口上。当需要附着在窗口上时，我们还可以通过 *IupExpander* 来动态地显示隐藏它。

C语言

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

#include <iup.h>
#include <iup_config.h>
#include <cd.h>
#include <cdprint.h>
#include <cdiup.h>
#include <im.h>
#include <im_image.h>
#include <im_convert.h>
#include <im_process.h>
#include <iupim.h>
#include <cdim.h>

//#define USE_OPENGL 1
//#define USE_CONTEXTPLUS 1

#ifdef USE_OPENGL
#include <iupgl.h>
#include <cdgl.h>
#endif

#if _MSC_VER < 1800 /* vc12 (2013) */
#define DEFINE_ROUND
#endif

#ifdef DEFINE_ROUND
double round(double x)
{
  return (int)(x>0? x+0.5: x-0.5);
}
#endif


/********************************** Images *************************************
```

```
Ihandle* load_image_PaintLine(void)
{
  unsigned char imgdata[] = {
    0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 255, 255, 255, 0, 255, 255, 255
    0, 0, 0, 2, 0, 0, 0, 5, 0, 0, 0, 12, 0, 0, 0, 0, 255, 255, 255, 0, 255, 255, 25
    0, 0, 0, 0, 0, 0, 0, 13, 0, 0, 0, 239, 0, 0, 0, 60, 255, 255, 255, 0, 255, 255,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 84, 0, 0, 0, 239, 0, 0, 0, 60, 255, 255, 255,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 84, 0, 0, 0, 239
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25

  Ihandle* image = IupImageRGBA(16, 16, imgdata);
  return image;
}

Ihandle* load_image_Pointer(void)
{
  unsigned char imgdata[] = {
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 162, 180, 203, 255, 162,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 162, 180, 203, 255, 162,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 162, 180, 203, 255, 240,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 162, 180, 203, 255, 255,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 158, 176, 200, 255, 255,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 153, 172, 195, 255, 255,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 148, 166, 189, 255, 255,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 141, 159, 183, 255, 254,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 134, 152, 176, 255, 249,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 127, 145, 170, 255, 241,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 119, 138, 163, 255, 203,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 112, 131, 156, 255, 82, 1
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 105, 124, 150, 255, 255,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
```

```
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25

  Ihandle* image = IupImageRGBA(16, 16, imgdata);
  return image;
}

Ihandle* load_image_PaintPencil(void)
{
  unsigned char imgdata[] = {
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 175, 15
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 175, 158, 120, 66, 179, 1
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 152, 157, 163, 255, 199,
    255, 255, 255, 0, 255, 255, 255, 0, 152, 157, 163, 118, 152, 157, 163, 255, 254
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 118, 110, 113, 117, 255, 81, 84, 8
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 224, 0, 0, 0, 118, 81, 84, 87, 104
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25

  Ihandle* image = IupImageRGBA(16, 16, imgdata);
  return image;
}

Ihandle* load_image_PaintColorPicker(void)
{
  unsigned char imgdata[] = {
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
```

```
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 152, 16
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 152, 16
    255, 255, 255, 0, 43, 134, 240, 46, 43, 134, 240, 107, 152, 166, 185, 255, 255,
    255, 255, 255, 0, 43, 133, 239, 170, 43, 133, 240, 255, 87, 105, 130, 255, 79,
    255, 255, 255, 0, 42, 120, 237, 46, 42, 124, 238, 107, 42, 125, 238, 161, 42, 1
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25

  Ihandle* image = IupImageRGBA(16, 16, imgdata);
  return image;
}

Ihandle* load_image_PaintEllipse(void)
{
  unsigned char imgdata[] = {
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 16, 0, 0, 0, 155
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 24, 0, 0, 0, 215, 0, 0, 0, 143, 0,
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 199, 0, 0, 0, 120, 255, 255, 255,
    255, 255, 255, 0, 0, 0, 0, 60, 0, 0, 0, 211, 255, 255, 255, 0, 255, 255, 255, 0
    255, 255, 255, 0, 0, 0, 0, 120, 0, 0, 0, 139, 255, 255, 255, 0, 255, 255, 255,
    255, 255, 255, 0, 0, 0, 0, 120, 0, 0, 0, 135, 255, 255, 255, 0, 255, 255, 255,
    255, 255, 255, 0, 0, 0, 0, 68, 0, 0, 0, 211, 255, 255, 255, 0, 255, 255, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 199, 0, 0, 0, 120, 255, 255, 255,
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 36, 0, 0, 0, 231, 0, 0, 0, 135, 0,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 20, 0, 0, 0, 163
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25

  Ihandle* image = IupImageRGBA(16, 16, imgdata);
  return image;
}

Ihandle* load_image_PaintRect(void)
{
  unsigned char imgdata[] = {
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
```

```
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 254, 0, 0, 0, 247, 0, 0, 0, 239, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255, 2
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255, 2
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255, 2
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255, 2
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255, 2
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255, 2
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255, 2
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255, 2
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25

  Ihandle* image = IupImageRGBA(16, 16, imgdata);
  return image;
}

Ihandle* load_image_PaintOval(void)
{
  unsigned char imgdata[] = {
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 80, 0, 0, 0, 239
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 48, 0, 0, 0, 251, 0, 0, 0, 255, 0,
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 183, 0, 0, 0, 255, 0, 0, 0, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 247, 0, 0, 0, 255, 0, 0, 0, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 247, 0, 0, 0, 255, 0, 0, 0, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 183, 0, 0, 0, 255, 0, 0, 0, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 56, 0, 0, 0, 251, 0, 0, 0, 255, 0,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 84, 0, 0, 0, 243
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25

  Ihandle* image = IupImageRGBA(16, 16, imgdata);
  return image;
}
```

```
Ihandle* load_image_PaintBox(void)
{
  unsigned char imgdata[] = {
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 254, 0, 0, 0, 247, 0, 0, 0, 239, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25

  Ihandle* image = IupImageRGBA(16, 16, imgdata);
  return image;
}

Ihandle* load_image_PaintZoomGrid(void)
{
  unsigned char imgdata[] = {
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 110, 155, 223, 255, 255,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 110, 155, 223, 255, 255,
    255, 255, 255, 0, 110, 155, 223, 255, 110, 155, 223, 255, 107, 152, 222, 255, 1
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 106, 149, 219, 255, 255,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 103, 148, 216, 255, 255,
    255, 255, 255, 0, 104, 151, 219, 255, 104, 148, 216, 255, 100, 145, 213, 255, 9
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 97, 142, 210, 255, 255, 2
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 93, 138, 206, 255, 255, 2
    255, 255, 255, 0, 95, 140, 212, 255, 92, 137, 207, 255, 88, 133, 203, 255, 84,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 84, 131, 199, 255, 255, 2
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 82, 125, 195, 255, 255, 2
    255, 255, 255, 0, 84, 129, 199, 255, 80, 125, 195, 255, 76, 121, 191, 255, 72,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 72, 117, 187, 255, 255, 2
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 68, 113, 185, 255, 255, 2
```

```
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25

  Ihandle* image = IupImageRGBA(16, 16, imgdata);
  return image;
}

Ihandle* load_image_PaintFill(void)
{
  unsigned char imgdata[] = {
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 50, 53, 53, 86, 59,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 52, 54, 54, 122, 72
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 77, 77, 76, 154, 14
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 124, 126, 129, 197,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 130, 141, 158, 45, 130, 143, 15
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 127, 148, 170, 24, 134, 155, 174, 172, 213,
    0, 0, 0, 0, 0, 0, 0, 0, 134, 147, 174, 19, 122, 146, 168, 179, 216, 223, 228, 2
    0, 0, 0, 0, 120, 134, 161, 19, 126, 147, 168, 157, 209, 217, 225, 245, 217, 231
    122, 132, 160, 27, 102, 124, 149, 172, 194, 209, 220, 241, 213, 228, 239, 246,
    125, 134, 152, 55, 111, 131, 154, 211, 189, 201, 214, 255, 214, 223, 234, 254,
    0, 255, 255, 1, 134, 140, 167, 38, 137, 155, 179, 206, 218, 223, 230, 254, 244,
    0, 0, 0, 0, 0, 0, 255, 1, 157, 173, 195, 47, 148, 166, 191, 205, 204, 210, 220,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 127, 142, 165, 34, 103, 124, 151, 197, 158,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 102, 110, 119, 30, 73, 96, 120,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 49, 82, 115, 31, 49
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 109, 11

  Ihandle* image = IupImageRGBA(16, 16, imgdata);
  return image;
}

Ihandle* load_image_PaintText(void)
{
  unsigned char imgdata[] = {
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 36, 0, 0, 0, 96, 0, 0, 0, 96, 0, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 96, 0, 0, 0, 255, 0, 0, 0, 203, 0,
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 96, 0, 0, 0, 163, 255, 255, 255, 0
    255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 96, 0, 0, 0, 48, 255, 255, 255, 0,
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
```

```
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
    255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25

  Ihandle* image = IupImageRGBA(16, 16, imgdata);
  return image;
}


/********************************* Utilities ************************************


const char* str_filetitle(const char *filename)
{
  /* Start at the last character */
  int len = (int)strlen(filename);
  int offset = len - 1;
  while (offset != 0)
  {
    if (filename[offset] == '\\' || filename[offset] == '/')
    {
      offset++;
      break;
    }
    offset--;
  }
  return filename + offset;
}

const char* str_fileext(const char *filename)
{
  /* Start at the last character */
  int len = (int)strlen(filename);
  int offset = len - 1;
  while (offset != 0)
  {
    if (filename[offset] == '\\' || filename[offset] == '/')
      break;
```

```
    if (filename[offset] == '.')
    {
      offset++;
      return filename + offset;
    }
    offset--;
  }
  return NULL;
}

int str_compare(const char *l, const char *r, int casesensitive)
{
  if (!l || !r)
    return 0;

  while (*l && *r)
  {
    int diff;
    char l_char = *l,
      r_char = *r;

    /* compute the difference of both characters */
    if (casesensitive)
      diff = l_char - r_char;
    else
      diff = tolower((int)l_char) - tolower((int)r_char);

    /* if they differ we have a result */
    if (diff != 0)
      return 0;

    /* otherwise process the next characters */
    ++l;
    ++r;
  }

  /* check also for terminator */
  if (*l == *r)
    return 1;

  if (*r == 0)
```

```
    return 1;  /* if second string is at terminator, then it is partially equal */

  return 0;
}

void show_error(const char* message, int is_error)
{
  Ihandle* dlg = IupMessageDlg();
  IupSetStrAttribute(dlg, "PARENTDIALOG", IupGetGlobal("PARENTDIALOG"));
  IupSetAttribute(dlg, "DIALOGTYPE", is_error ? "ERROR" : "WARNING");
  IupSetAttribute(dlg, "BUTTONS", "OK");
  IupSetStrAttribute(dlg, "TITLE", is_error ? "Error" : "Warning");
  IupSetStrAttribute(dlg, "VALUE", message);
  IupPopup(dlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
  IupDestroy(dlg);
}

void show_file_error(int error)
{
  switch (error)
  {
  case IM_ERR_OPEN:
    show_error("Error␣Opening␣File.", 1);
    break;
  case IM_ERR_MEM:
    show_error("Insufficient␣memory.", 1);
    break;
  case IM_ERR_ACCESS:
    show_error("Error␣Accessing␣File.", 1);
    break;
  case IM_ERR_DATA:
    show_error("Image␣type␣not␣Supported.", 1);
    break;
  case IM_ERR_FORMAT:
    show_error("Invalid␣Format.", 1);
    break;
  case IM_ERR_COMPRESS:
    show_error("Invalid␣or␣unsupported␣compression.", 1);
    break;
  default:
    show_error("Unknown␣Error.", 1);
  }
```

```
}

imImage* read_file(const char* filename)
{
  int error;
  imImage* image = imFileImageLoadBitmap(filename, 0, &error);
  if (error)
    show_file_error(error);
  return image;
}

int write_file(const char* filename, const imImage* image)
{
  const char* format = imImageGetAttribString(image, "FileFormat");
  int error = imFileImageSave(filename, format, image);
  if (error)
  {
    show_file_error(error);
    return 0;
  }
  return 1;
}

/* extracted from the SCROLLBAR attribute documentation */
void scroll_update(Ihandle* ih, int view_width, int view_height)
{
  /* view_width and view_height is the virtual space size */
  /* here we assume XMIN=0, XMAX=1, YMIN=0, YMAX=1 */
  int elem_width, elem_height;
  int canvas_width, canvas_height;
  int scrollbar_size = IupGetInt(NULL, "SCROLLBARSIZE");
  int border = IupGetInt(ih, "BORDER");

  IupGetIntInt(ih, "RASTERSIZE", &elem_width, &elem_height);

  /* if view is greater than canvas in one direction,
  then it has scrollbars,
  but this affects the opposite direction */
  elem_width -= 2 * border;  /* remove BORDER (always size=1) */
  elem_height -= 2 * border;
  canvas_width = elem_width;
  canvas_height = elem_height;
```

```
  if (view_width > elem_width)  /* check for horizontal scrollbar */
    canvas_height -= scrollbar_size;  /* affect vertical size */
  if (view_height > elem_height)
    canvas_width -= scrollbar_size;
  if (view_width <= elem_width && view_width > canvas_width)  /* check if still has
    canvas_height -= scrollbar_size;
  if (view_height <= elem_height && view_height > canvas_height)
    canvas_width -= scrollbar_size;
  if (canvas_width < 0) canvas_width = 0;
  if (canvas_height < 0) canvas_height = 0;

  IupSetFloat(ih, "DX", (float)canvas_width / (float)view_width);
  IupSetFloat(ih, "DY", (float)canvas_height / (float)view_height);
}

void scroll_calc_center(Ihandle* ih, float *x, float *y)
{
  *x = IupGetFloat(ih, "POSX") + IupGetFloat(ih, "DX") / 2.0f;
  *y = IupGetFloat(ih, "POSY") + IupGetFloat(ih, "DY") / 2.0f;
}

void scroll_center(Ihandle* ih, float old_center_x, float old_center_y)
{
  /* always update the scroll position
     keeping it proportional to the old position
     relative to the center of the ih. */

  float dx = IupGetFloat(ih, "DX");
  float dy = IupGetFloat(ih, "DY");

  float posx = old_center_x - dx / 2.0f;
  float posy = old_center_y - dy / 2.0f;

  if (posx < 0) posx = 0;
  if (posx > 1 - dx) posx = 1 - dx;

  if (posy < 0) posy = 0;
  if (posy > 1 - dy) posy = 1 - dy;

  IupSetFloat(ih, "POSX", posx);
  IupSetFloat(ih, "POSY", posy);
}
```

```
void scroll_move(Ihandle* ih, int canvas_width, int canvas_height, int move_x, int
{
  float posy = 0;
  float posx = 0;

  if (move_x == 0 && move_y == 0)
    return;

  if (canvas_height < view_height)
  {
    posy = IupGetFloat(ih, "POSY");
    posy -= (float)move_y / (float)view_height;
  }

  if (canvas_width < view_width)
  {
    posx = IupGetFloat(ih, "POSX");
    posx -= (float)move_x / (float)view_width;
  }

  if (posx != 0 || posy != 0)
  {
    IupSetFloat(ih, "POSX", posx);
    IupSetFloat(ih, "POSY", posy);
    IupUpdate(ih);
  }
}

void zoom_update(Ihandle* ih, double zoom_index)
{
  Ihandle* zoom_lbl = IupGetDialogChild(ih, "ZOOMLABEL");
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  double zoom_factor = pow(2, zoom_index);
  IupSetStrf(zoom_lbl, "TITLE", "%.0f%%", floor(zoom_factor * 100));
  if (image)
  {
    float old_center_x, old_center_y;
    int view_width = (int)(zoom_factor * image->width);
    int view_height = (int)(zoom_factor * image->height);
```

```
    scroll_calc_center(canvas, &old_center_x, &old_center_y);

    scroll_update(canvas, view_width, view_height);

    scroll_center(canvas, old_center_x, old_center_y);
  }
  IupUpdate(canvas);
}

void image_flood_fill(imImage* image, int start_x, int start_y, long replace_color,
{
  float color[3];
  double tol;

  color[0] = (float)cdRed(replace_color);
  color[1] = (float)cdGreen(replace_color);
  color[2] = (float)cdBlue(replace_color);

  /* max value = 255*255*3 = 195075 */
  /* sqrt(195075) = 441 */
  tol = (441 * tol_percent) / 100;

  /* still too high */
  tol = tol / 5;  /* empirical reduce. TODO: What is the best formula? */

  imProcessRenderFloodFill(image, start_x, start_y, color, (float)tol);
}

void image_fill_white(imImage* image)
{
  float color[3];

  color[0] = 255;
  color[1] = 255;
  color[2] = 255;

  imProcessRenderConstant(image, color);
}

void update_image(Ihandle* canvas, imImage* image, int update_size)
{
  imImage* old_image = (imImage*)IupGetAttribute(canvas, "IMAGE");
```

```
  IupSetAttribute(canvas, "DIRTY", "Yes");
  IupSetAttribute(canvas, "IMAGE", (char*)image);

  if (old_image)
    imImageDestroy(old_image);

  if (update_size)
  {
    Ihandle* size_lbl = IupGetDialogChild(canvas, "SIZELABEL");
    Ihandle* zoom_val = IupGetDialogChild(canvas, "ZOOMVAL");
    double zoom_index = IupGetDouble(zoom_val, "VALUE");
    IupSetfAttribute(size_lbl, "TITLE", "%d x %d px", image->width, image->height);
    zoom_update(canvas, zoom_index);
  }
  else
    IupUpdate(canvas);
}

void set_new_image(Ihandle* canvas, imImage* image, const char* filename, int dirty
{
  imImage* old_image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  Ihandle* size_lbl = IupGetDialogChild(canvas, "SIZELABEL");
  Ihandle* zoom_val = IupGetDialogChild(canvas, "ZOOMVAL");
  const char* format;

  if (filename)
  {
    IupSetStrAttribute(canvas, "FILENAME", filename);
    IupSetfAttribute(IupGetDialog(canvas), "TITLE", "%s - Simple Paint", str_fileti
  }
  else
  {
    IupSetAttribute(canvas, "FILENAME", NULL);
    IupSetAttribute(IupGetDialog(canvas), "TITLE", "Untitled - Simple Paint");
  }

  /* we are going to support only RGB images with no alpha */
  imImageRemoveAlpha(image);
  if (image->color_space != IM_RGB)
  {
    imImage* new_image = imImageCreateBased(image, -1, -1, IM_RGB, -1);
```

```
    imConvertColorSpace(image, new_image);
    imImageDestroy(image);

    image = new_image;
  }

  /* default file format */
  format = imImageGetAttribString(image, "FileFormat");
  if (!format)
    imImageSetAttribString(image, "FileFormat", "JPEG");

  IupSetAttribute(canvas, "DIRTY", dirty ? "Yes" : "No");
  IupSetAttribute(canvas, "IMAGE", (char*)image);

  IupSetfAttribute(size_lbl, "TITLE", "%d x %d px", image->width, image->height);

  if (old_image)
    imImageDestroy(old_image);

  IupSetDouble(zoom_val, "VALUE", 0);
  zoom_update(canvas, 0);
}

void check_new_file(Ihandle* dlg)
{
  Ihandle* canvas = IupGetDialogChild(dlg, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  if (!image)
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
    int width = IupConfigGetVariableIntDef(config, "NewImage", "Width", 640);
    int height = IupConfigGetVariableIntDef(config, "NewImage", "Height", 480);

    image = imImageCreate(width, height, IM_RGB, IM_BYTE);
    if (!image)
    {
      show_file_error(IM_ERR_MEM);
      return;
    }

    image_fill_white(image);
```

```
    set_new_image(canvas, image, NULL, 0);
  }
}

void open_file(Ihandle* ih, const char* filename)
{
  imImage* image = read_file(filename);
  if (image)
  {
    Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");

    set_new_image(canvas, image, filename, 0);

    IupConfigRecentUpdate(config, filename);
  }
}

void save_file(Ihandle* canvas)
{
  char* filename = IupGetAttribute(canvas, "FILENAME");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  if (write_file(filename, image))
    IupSetAttribute(canvas, "DIRTY", "NO");
}

void set_file_format(imImage* image, const char* filename)
{
  const char* ext = str_fileext(filename);
  const char* format = "JPEG";
  if (str_compare(ext, "jpg", 0) || str_compare(ext, "jpeg", 0))
    format = "JPEG";
  else if (str_compare(ext, "bmp", 0))
    format = "BMP";
  else if (str_compare(ext, "png", 0))
    format = "PNG";
  else if (str_compare(ext, "tga", 0))
    format = "TGA";
  else if (str_compare(ext, "tif", 0) || str_compare(ext, "tiff", 0))
    format = "TIFF";
  imImageSetAttribString(image, "FileFormat", format);
}
```

```
void saveas_file(Ihandle* canvas, const char* filename)
{
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");

  set_file_format(image, filename);

  if (write_file(filename, image))
  {
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");

    IupSetfAttribute(IupGetDialog(canvas), "TITLE", "%s - Simple Paint", str_fileti
    IupSetStrAttribute(canvas, "FILENAME", filename);
    IupSetAttribute(canvas, "DIRTY", "NO");

    IupConfigRecentUpdate(config, filename);
  }
}

int save_check(Ihandle* ih)
{
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  if (IupGetInt(canvas, "DIRTY"))
  {
    switch (IupAlarm("Warning", "File not saved! Save it now?", "Yes", "No", "Cance
    {
    case 1:  /* save the changes and continue */
      save_file(canvas);
      break;
    case 2:  /* ignore the changes and continue */
      break;
    case 3:  /* cancel */
      return 0;
    }
  }
  return 1;
}

void toggle_bar_visibility(Ihandle* item, Ihandle* ih)
{
  if (IupGetInt(item, "VALUE"))
  {
```

```
    IupSetAttribute(ih, "FLOATING", "YES");
    IupSetAttribute(ih, "VISIBLE", "NO");
    IupSetAttribute(item, "VALUE", "OFF");
  }
  else
  {
    IupSetAttribute(ih, "FLOATING", "NO");
    IupSetAttribute(ih, "VISIBLE", "YES");
    IupSetAttribute(item, "VALUE", "ON");
  }

  IupRefresh(ih);  /* refresh the dialog layout */
}

int select_file(Ihandle* parent_dlg, int is_open)
{
  Ihandle* config = (Ihandle*)IupGetAttribute(parent_dlg, "CONFIG");
  Ihandle* canvas = IupGetDialogChild(parent_dlg, "CANVAS");
  const char* dir = IupConfigGetVariableStr(config, "MainWindow", "LastDirectory");

  Ihandle* filedlg = IupFileDlg();
  if (is_open)
    IupSetAttribute(filedlg, "DIALOGTYPE", "OPEN");
  else
  {
    IupSetAttribute(filedlg, "DIALOGTYPE", "SAVE");
    IupSetStrAttribute(filedlg, "FILE", IupGetAttribute(canvas, "FILENAME"));
  }
  IupSetAttribute(filedlg, "EXTFILTER", "Image␣Files|*.bmp;*.jpg;*.png;*.tif;*.tga|
  IupSetStrAttribute(filedlg, "DIRECTORY", dir);
  IupSetAttributeHandle(filedlg, "PARENTDIALOG", parent_dlg);

  IupPopup(filedlg, IUP_CENTERPARENT, IUP_CENTERPARENT);
  if (IupGetInt(filedlg, "STATUS") != -1)
  {
    char* filename = IupGetAttribute(filedlg, "VALUE");
    if (is_open)
      open_file(parent_dlg, filename);
    else
      saveas_file(canvas, filename);

    dir = IupGetAttribute(filedlg, "DIRECTORY");
```

```
    IupConfigSetVariableStr(config, "MainWindow", "LastDirectory", dir);
  }

  IupDestroy(filedlg);
  return IUP_DEFAULT;
}

void view_fit_rect(int canvas_width, int canvas_height, int image_width, int image_
{
  *view_width = canvas_width;
  *view_height = (canvas_width * image_height) / image_width;

  if (*view_height > canvas_height)
  {
    *view_height = canvas_height;
    *view_width = (canvas_height * image_width) / image_height;
  }
}

double view_zoom_rect(Ihandle* ih, int image_width, int image_height, int *_x, int
{
  int x, y, canvas_width, canvas_height;
  int view_width, view_height;
  Ihandle* zoom_val = IupGetDialogChild(ih, "ZOOMVAL");
  double zoom_index = IupGetDouble(zoom_val, "VALUE");
  double zoom_factor = pow(2, zoom_index);

  float posy = IupGetFloat(ih, "POSY");
  float posx = IupGetFloat(ih, "POSX");

  IupGetIntInt(ih, "DRAWSIZE", &canvas_width, &canvas_height);

  view_width = (int)(zoom_factor * image_width);
  view_height = (int)(zoom_factor * image_height);

  if (canvas_width < view_width)
    x = (int)floor(-posx*view_width);
  else
    x = (canvas_width - view_width) / 2;

  if (canvas_height < view_height)
  {
```

```
    /* posy is top-bottom, CD is bottom-top.
    invert posy reference (YMAX-DY - POSY) */
    float dy = IupGetFloat(ih, "DY");
    posy = 1.0f - dy - posy;
    y = (int)floor(-posy*view_height);
  }
  else
    y = (canvas_height - view_height) / 2;

  *_x = x;
  *_y = y;
  *_view_width = view_width;
  *_view_height = view_height;

  return zoom_factor;
}

void view_zoom_offset(int view_x, int view_y, int image_width, int image_height, do
{
  *x -= view_x;
  *y -= view_y;

  *x = (int)(*x / zoom_factor);
  *y = (int)(*y / zoom_factor);

  if (*x < 0) *x = 0;
  if (*y < 0) *y = 0;
  if (*x > image_width - 1) *x = image_width - 1;
  if (*y > image_height - 1) *y = image_height - 1;
}

int tool_get_text_enter_cb(void)
{
  return IUP_CLOSE;
}

void tool_get_text(Ihandle* toolbox)
{
  Ihandle *text, *dlg;

  char* value = IupGetAttribute(toolbox, "TOOLTEXT");
  char* font = IupGetAttribute(toolbox, "TOOLFONT");
```

```
  text = IupText(NULL);
  IupSetAttribute(text, "EXPAND", "YES");
  IupSetStrAttribute(text, "VALUE", value);
  IupSetStrAttribute(text, "FONT", font);
  IupSetAttribute(text, "VISIBLECOLUMNS", "20");

  dlg = IupDialog(text);

  IupSetStrAttribute(dlg, "TITLE", "Enter Text:");
  IupSetAttribute(dlg, "MINBOX", "NO");
  IupSetAttribute(dlg, "MAXBOX", "NO");
  IupSetCallback(dlg, "K_CR", (Icallback)tool_get_text_enter_cb);
  IupSetAttributeHandle(dlg, "PARENTDIALOG", toolbox);

  IupPopup(dlg, IUP_MOUSEPOS, IUP_MOUSEPOS);

  value = IupGetAttribute(text, "VALUE");
  IupSetStrAttribute(toolbox, "TOOLTEXT", value);

  IupDestroy(dlg);
}

void tool_draw_pencil(Ihandle* toolbox, imImage* image, int start_x, int start_y, i
{
  double res = IupGetDouble(NULL, "SCREENDPI") / 25.4;
  unsigned char r, g, b;
  cdCanvas* rgb_canvas;

  int line_width = IupGetInt(toolbox, "TOOLWIDTH");
  IupGetRGB(toolbox, "TOOLCOLOR", &r, &g, &b);

  /* do not use line style here */
  rgb_canvas = cdCreateCanvas(CD_IMIMAGE, image);
  cdCanvasSetfAttribute(rgb_canvas, "RESOLUTION", "%g", res);
  cdCanvasForeground(rgb_canvas, cdEncodeColor(r, g, b));
  cdCanvasLineWidth(rgb_canvas, line_width);
  cdCanvasLine(rgb_canvas, start_x, start_y, end_x, end_y);
  cdKillCanvas(rgb_canvas);
}

void tool_draw_overlay(Ihandle* toolbox, cdCanvas* cd_canvas, int start_x, int start
```

```
{
  int tool_index = IupGetInt(toolbox, "TOOLINDEX");
  int line_width = IupGetInt(toolbox, "TOOLWIDTH");
  int line_style = IupGetInt(toolbox, "TOOLSTYLE") - 1;
  unsigned char r, g, b;
  IupGetRGB(toolbox, "TOOLCOLOR", &r, &g, &b);

  cdCanvasForeground(cd_canvas, cdEncodeColor(r, g, b));
  cdCanvasLineWidth(cd_canvas, line_width);
  if (line_width == 1)
    cdCanvasLineStyle(cd_canvas, line_style);

  if (tool_index == 3)  /* Line */
    cdCanvasLine(cd_canvas, start_x, start_y, end_x, end_y);
  else if (tool_index == 4)  /* Rect */
    cdCanvasRect(cd_canvas, start_x, end_x, start_y, end_y);
  else if (tool_index == 5)  /* Box */
    cdCanvasBox(cd_canvas, start_x, end_x, start_y, end_y);
  else if (tool_index == 6)  /* Ellipse */
    cdCanvasArc(cd_canvas, (end_x + start_x) / 2, (end_y + start_y) / 2, abs(end_x
  else if (tool_index == 7)  /* Oval */
    cdCanvasSector(cd_canvas, (end_x + start_x) / 2, (end_y + start_y) / 2, abs(end_
  else if (tool_index == 8)  /* Text */
  {
    cdCanvasTextAlignment(cd_canvas, CD_SOUTH_WEST);
    cdCanvasNativeFont(cd_canvas, IupGetAttribute(toolbox, "TOOLFONT"));
    cdCanvasText(cd_canvas, end_x, end_y, IupGetAttribute(toolbox, "TOOLTEXT"));
  }
}


/********************************** Callbacks ************************************


int canvas_action_cb(Ihandle* canvas)
{
  unsigned int ri, gi, bi;
  imImage* image;
  cdCanvas* cd_canvas = (cdCanvas*)IupGetAttribute(canvas, "cdCanvas");
  Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
  const char* background = IupConfigGetVariableStrDef(config, "Canvas", "Background
```

```
#ifdef USE_OPENGL
  IupGLMakeCurrent(canvas);
#endif
  cdCanvasActivate(cd_canvas);

  /* draw the background */
  sscanf(background, "%u %u %u", &ri, &gi, &bi);
  cdCanvasBackground(cd_canvas, cdEncodeColor((unsigned char)ri, (unsigned char)gi,
  cdCanvasClear(cd_canvas);

  /* draw the image at the center of the canvas */
  image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  if (image)
  {
    int x, y, view_width, view_height;
    view_zoom_rect(canvas, image->width, image->height, &x, &y, &view_width, &view_

    /* black line around the image */
    cdCanvasForeground(cd_canvas, CD_BLACK);
    cdCanvasLineWidth(cd_canvas, 1);
    cdCanvasLineStyle(cd_canvas, CD_CONTINUOUS);
    cdCanvasRect(cd_canvas, x - 1, x + view_width, y - 1, y + view_height);

    /* Some CD drivers have interpolation options for image zoom */
    /* we force NEAREST so we can see the pixel boundary in zoom in */
    /* an alternative would be to set BILINEAR when zoom out */
    cdCanvasSetAttribute(cd_canvas, "IMGINTERP", "NEAREST");  /* affects only drive
    cdCanvasPutImImage(cd_canvas, image, x, y, view_width, view_height);

    if (IupConfigGetVariableInt(config, "Canvas", "ZoomGrid"))
    {
      Ihandle* zoom_val = IupGetDialogChild(canvas, "ZOOMVAL");
      double zoom_index = IupGetDouble(zoom_val, "VALUE");
      if (zoom_index > 1)
      {
        int ix, iy;
        double zoom_factor = pow(2, zoom_index);

        cdCanvasForeground(cd_canvas, CD_GRAY);

        for (ix = 0; ix < image->width; ix++)
        {
```

```
          int gx = (int)(ix * zoom_factor);
          cdCanvasLine(cd_canvas, gx + x, y, gx + x, y + view_height);
        }
        for (iy = 0; iy < image->height; iy++)
        {
          int gy = (int)(iy * zoom_factor);
          cdCanvasLine(cd_canvas, x, gy + y, x + view_width, gy + y);
        }
      }
    }

    if (IupGetAttribute(canvas, "OVERLAY"))
    {
      Ihandle* toolbox = (Ihandle*)IupGetAttribute(canvas, "TOOLBOX");
      int start_x = IupGetInt(canvas, "START_X");
      int start_y = IupGetInt(canvas, "START_Y");
      int end_x = IupGetInt(canvas, "END_X");
      int end_y = IupGetInt(canvas, "END_Y");

      double scale_x = (double)view_width / (double)image->width;
      double scale_y = (double)view_height / (double)image->height;

      /* offset and scale drawing in screen to macth the image */
      if (scale_x > 1 || scale_y > 1)
      {
        /* also draw at the center of the pixel when zoom in */
        cdCanvasTransformTranslate(cd_canvas, x + scale_x / 2, y + scale_y / 2);
      }
      else
        cdCanvasTransformTranslate(cd_canvas, x, y);
      cdCanvasTransformScale(cd_canvas, scale_x, scale_y);

      tool_draw_overlay(toolbox, cd_canvas, start_x, start_y, end_x, end_y);

      cdCanvasTransform(cd_canvas, NULL);
    }
  }

  cdCanvasFlush(cd_canvas);

#ifdef USE_OPENGL
  IupGLSwapBuffers(canvas);
```

```
#endif
  return IUP_DEFAULT;
}

int canvas_map_cb(Ihandle* canvas)
{
  cdCanvas* cd_canvas;

#ifdef USE_OPENGL
  double res = IupGetDouble(NULL, "SCREENDPI") / 25.4;
  IupGLMakeCurrent(canvas);
  cd_canvas = cdCreateCanvasf(CD_GL, "10x10 %g", res);
#else
#ifdef USE_CONTEXTPLUS
  cdUseContextPlus(1);
#endif
  cd_canvas = cdCreateCanvas(CD_IUPDBUFFER, canvas);
#ifdef USE_CONTEXTPLUS
  cdUseContextPlus(0);
#endif
#endif

  IupSetAttribute(canvas, "cdCanvas", (char*)cd_canvas);
  return IUP_DEFAULT;
}

int canvas_unmap_cb(Ihandle* canvas)
{
  cdCanvas* cd_canvas = (cdCanvas*)IupGetAttribute(canvas, "cdCanvas");
  cdKillCanvas(cd_canvas);
  return IUP_DEFAULT;
}

int zoomout_action_cb(Ihandle* ih)
{
  Ihandle* zoom_val = IupGetDialogChild(ih, "ZOOMVAL");
  double zoom_index = IupGetDouble(zoom_val, "VALUE");
  zoom_index--;
  if (zoom_index < -6)
    zoom_index = -6;
  IupSetDouble(zoom_val, "VALUE", round(zoom_index));  /* fixed increments when usi
```

```
  zoom_update(ih, zoom_index);
  return IUP_DEFAULT;
}

int zoomin_action_cb(Ihandle* ih)
{
  Ihandle* zoom_val = IupGetDialogChild(ih, "ZOOMVAL");
  double zoom_index = IupGetDouble(zoom_val, "VALUE");
  zoom_index++;
  if (zoom_index > 6)
    zoom_index = 6;
  IupSetDouble(zoom_val, "VALUE", round(zoom_index));  /* fixed increments when usi

  zoom_update(ih, zoom_index);
  return IUP_DEFAULT;
}

int actualsize_action_cb(Ihandle* ih)
{
  Ihandle* zoom_val = IupGetDialogChild(ih, "ZOOMVAL");
  IupSetDouble(zoom_val, "VALUE", 0);
  zoom_update(ih, 0);
  return IUP_DEFAULT;
}

int canvas_resize_cb(Ihandle* canvas)
{
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  if (image)
  {
    Ihandle* zoom_val = IupGetDialogChild(canvas, "ZOOMVAL");
    double zoom_index = IupGetDouble(zoom_val, "VALUE");
    double zoom_factor = pow(2, zoom_index);
    float old_center_x, old_center_y;

    int view_width = (int)(zoom_factor * image->width);
    int view_height = (int)(zoom_factor * image->height);

    scroll_calc_center(canvas, &old_center_x, &old_center_y);

    scroll_update(canvas, view_width, view_height);
```

```
    scroll_center(canvas, old_center_x, old_center_y);
  }

#ifdef USE_OPENGL
  {
    int canvas_width, canvas_height;
    double res = IupGetDouble(NULL, "SCREENDPI") / 25.4;
    cdCanvas* cd_canvas = (cdCanvas*)IupGetAttribute(canvas, "cdCanvas");
    IupGetIntInt(canvas, "DRAWSIZE", &canvas_width, &canvas_height);

    IupGLMakeCurrent(canvas);
    cdCanvasSetfAttribute(cd_canvas, "SIZE", "%dx%d %g", canvas_width, canvas_heigh
  }
#endif
  return IUP_DEFAULT;
}

int canvas_wheel_cb(Ihandle* canvas, float delta)
{
  if (IupGetInt(NULL, "CONTROLKEY"))
  {
    if (delta < 0)
      zoomout_action_cb(canvas);
    else
      zoomin_action_cb(canvas);
  }
  else
  {
    float posy = IupGetFloat(canvas, "POSY");
    posy -= delta * IupGetFloat(canvas, "DY") / 10.0f;
    IupSetFloat(canvas, "POSY", posy);
    IupUpdate(canvas);
  }
  return IUP_DEFAULT;
}

int canvas_button_cb(Ihandle* canvas, int button, int pressed, int x, int y)
{
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  if (image)
  {
    int cursor_x = x, cursor_y = y;
```

```
int view_x, view_y, view_width, view_height;
double zoom_factor = view_zoom_rect(canvas, image->width, image->height, &view_

/* y is top-down in IUP */
int canvas_height = IupGetInt2(canvas, "DRAWSIZE");
y = canvas_height - y - 1;

/* inside image area */
if (x > view_x && y > view_y && x < view_x + view_width && y < view_y + view_he
{
  view_zoom_offset(view_x, view_y, image->width, image->height, zoom_factor, &x

  if (button == IUP_BUTTON1)
  {
    if (pressed)
    {
      IupSetInt(canvas, "START_X", x);
      IupSetInt(canvas, "START_Y", y);
      IupSetInt(canvas, "START_CURSOR_X", cursor_x);
      IupSetInt(canvas, "START_CURSOR_Y", cursor_y);
    }
    else
    {
      Ihandle* toolbox = (Ihandle*)IupGetAttribute(canvas, "TOOLBOX");
      int tool_index = IupGetInt(toolbox, "TOOLINDEX");

      if (tool_index == 1)  /* Color Picker */
      {
        Ihandle* color = IupGetDialogChild(toolbox, "COLOR");
        unsigned char** data = (unsigned char**)image->data;
        unsigned char r, g, b;
        int offset;

        offset = y * image->width + x;
        r = data[0][offset];
        g = data[1][offset];
        b = data[2][offset];

        IupSetRGB(color, "BGCOLOR", r, g, b);
        IupSetRGB(toolbox, "TOOLCOLOR", r, g, b);
      }
      else if (tool_index == 2)  /* Pencil */
```

```
  {
    int start_x = IupGetInt(canvas, "START_X");
    int start_y = IupGetInt(canvas, "START_Y");

    tool_draw_pencil(toolbox, image, start_x, start_y, x, y);

    IupSetAttribute(canvas, "DIRTY", "Yes");

    IupUpdate(canvas);

    IupSetInt(canvas, "START_X", x);
    IupSetInt(canvas, "START_Y", y);
  }
  else if (tool_index >= 3 && tool_index <= 8)  /* Shapes */
  {
    if (IupGetAttribute(canvas, "OVERLAY"))
    {
      int start_x = IupGetInt(canvas, "START_X");
      int start_y = IupGetInt(canvas, "START_Y");
      double res = IupGetDouble(NULL, "SCREENDPI") / 25.4;

      cdCanvas* rgb_canvas = cdCreateCanvas(CD_IMIMAGE, image);
      cdCanvasSetfAttribute(rgb_canvas, "RESOLUTION", "%g", res);

      tool_draw_overlay(toolbox, rgb_canvas, start_x, start_y, x, y);

      cdKillCanvas(rgb_canvas);

      IupSetAttribute(canvas, "OVERLAY", NULL);
      IupSetAttribute(canvas, "DIRTY", "Yes");

      IupUpdate(canvas);
    }
  }
  else if (tool_index == 9)  /* Fill Color */
  {
    double tol_percent = IupGetDouble(toolbox, "TOOLFILLTOL");
    unsigned char r, g, b;
    IupGetRGB(toolbox, "TOOLCOLOR", &r, &g, &b);

    image_flood_fill(image, x, y, cdEncodeColor(r, g, b), tol_percent);
    IupSetAttribute(canvas, "DIRTY", "Yes");
```

```
            IupUpdate(canvas);
          }
        }
      }
      else if (button == IUP_BUTTON3)
      {
        if (!pressed)
        {
          Ihandle* toolbox = (Ihandle*)IupGetAttribute(canvas, "TOOLBOX");
          int tool_index = IupGetInt(toolbox, "TOOLINDEX");
          if (tool_index == 8)  /* Text */
            tool_get_text(toolbox);
        }
      }
    }
  }

  return IUP_DEFAULT;
}

int canvas_motion_cb(Ihandle* canvas, int x, int y, char *status)
{
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  if (image)
  {
    int cursor_x = x, cursor_y = y;
    int view_x, view_y, view_width, view_height;
    double zoom_factor = view_zoom_rect(canvas, image->width, image->height, &view_

    /* y is top-down in IUP */
    int canvas_height = IupGetInt2(canvas, "DRAWSIZE");
    y = canvas_height - y - 1;

    /* inside image area */
    if (x > view_x && y > view_y && x < view_x + view_width && y < view_y + view_he
    {
      Ihandle* status_lbl = IupGetDialogChild(canvas, "STATUSLABEL");
      unsigned char** data = (unsigned char**)image->data;
      unsigned char r, g, b;
      int offset;
```

```
view_zoom_offset(view_x, view_y, image->width, image->height, zoom_factor, &x

offset = y * image->width + x;
r = data[0][offset];
g = data[1][offset];
b = data[2][offset];

IupSetStrf(status_lbl, "TITLE", "(%4d, %4d) = %3d %3d %3d", x, y, (int)r, (in

if (iup_isbutton1(status)) /* button1 is pressed */
{
  Ihandle* toolbox = (Ihandle*)IupGetAttribute(canvas, "TOOLBOX");
  int tool_index = IupGetInt(toolbox, "TOOLINDEX");

  if (tool_index == 0)  /* Pointer */
  {
    int start_cursor_x = IupGetInt(canvas, "START_CURSOR_X");
    int start_cursor_y = IupGetInt(canvas, "START_CURSOR_Y");

    int canvas_width = IupGetInt(canvas, "DRAWSIZE");

    scroll_move(canvas, canvas_width, canvas_height, cursor_x - start_cursor_x

    IupSetInt(canvas, "START_CURSOR_X", cursor_x);
    IupSetInt(canvas, "START_CURSOR_Y", cursor_y);
  }
  else if (tool_index == 2)  /* Pencil */
  {
    int start_x = IupGetInt(canvas, "START_X");
    int start_y = IupGetInt(canvas, "START_Y");

    tool_draw_pencil(toolbox, image, start_x, start_y, x, y);

    IupSetAttribute(canvas, "DIRTY", "Yes");

    IupUpdate(canvas);

    IupSetInt(canvas, "START_X", x);
    IupSetInt(canvas, "START_Y", y);
  }
  else if (tool_index >= 3 && tool_index <= 8)  /* Shapes */
  {
```

```
          IupSetInt(canvas, "END_X", x);
          IupSetInt(canvas, "END_Y", y);
          IupSetAttribute(canvas, "OVERLAY", "Yes");
          IupUpdate(canvas);
        }
      }
    }
  }

  return IUP_DEFAULT;
}

int zoom_valuechanged_cb(Ihandle* val)
{
  double zoom_index = IupGetDouble(val, "VALUE");
  zoom_update(val, zoom_index);
  return IUP_DEFAULT;
}

int dropfiles_cb(Ihandle* ih, const char* filename)
{
  if (save_check(ih))
    open_file(ih, filename);

  return IUP_DEFAULT;
}

int file_menu_open_cb(Ihandle* ih)
{
  Ihandle* item_revert = IupGetDialogChild(ih, "ITEM_REVERT");
  Ihandle* item_save = IupGetDialogChild(ih, "ITEM_SAVE");
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  char* filename = IupGetAttribute(canvas, "FILENAME");
  int dirty = IupGetInt(canvas, "DIRTY");

  if (dirty)
    IupSetAttribute(item_save, "ACTIVE", "YES");
  else
    IupSetAttribute(item_save, "ACTIVE", "NO");

  if (dirty && filename)
    IupSetAttribute(item_revert, "ACTIVE", "YES");
```

```
  else
    IupSetAttribute(item_revert, "ACTIVE", "NO");
  return IUP_DEFAULT;
}

int edit_menu_open_cb(Ihandle* ih)
{
  Ihandle *clipboard = IupClipboard();

  Ihandle *item_paste = IupGetDialogChild(ih, "ITEM_PASTE");

  if (!IupGetInt(clipboard, "IMAGEAVAILABLE"))
    IupSetAttribute(item_paste, "ACTIVE", "NO");
  else
    IupSetAttribute(item_paste, "ACTIVE", "YES");

  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int config_recent_cb(Ihandle* ih)
{
  if (save_check(ih))
  {
    char* filename = IupGetAttribute(ih, "TITLE");
    open_file(ih, filename);
  }
  return IUP_DEFAULT;
}

int item_new_action_cb(Ihandle* item_new)
{
  if (save_check(item_new))
  {
    Ihandle* canvas = IupGetDialogChild(item_new, "CANVAS");
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
    int width = IupConfigGetVariableIntDef(config, "NewImage", "Width", 640);
    int height = IupConfigGetVariableIntDef(config, "NewImage", "Height", 480);

    if (IupGetParam("New␣Image", NULL, NULL, "Width:␣%i[1,]\nHeight:␣%i[1,]\n", &wi
    {
      imImage* image = imImageCreate(width, height, IM_RGB, IM_BYTE);
```

```
      if (!image)
      {
        show_file_error(IM_ERR_MEM);
        return IUP_DEFAULT;
      }

      image_fill_white(image);

      IupConfigSetVariableInt(config, "NewImage", "Width", width);
      IupConfigSetVariableInt(config, "NewImage", "Height", height);

      set_new_image(canvas, image, NULL, 0);
    }
  }

  return IUP_DEFAULT;
}

int item_open_action_cb(Ihandle* item_open)
{
  if (!save_check(item_open))
    return IUP_DEFAULT;

  return select_file(IupGetDialog(item_open), 1);
}

int item_saveas_action_cb(Ihandle* item_saveas)
{
  return select_file(IupGetDialog(item_saveas), 0);
}

int item_save_action_cb(Ihandle* item_save)
{
  Ihandle* canvas = IupGetDialogChild(item_save, "CANVAS");
  char* filename = IupGetAttribute(canvas, "FILENAME");
  if (!filename)
    item_saveas_action_cb(item_save);
  else
  {
    /* test again because in can be called using the hot key */
    int dirty = IupGetInt(canvas, "DIRTY");
    if (dirty)
```

```
      save_file(canvas);
  }
  return IUP_DEFAULT;
}

int item_revert_action_cb(Ihandle* item_revert)
{
  Ihandle* canvas = IupGetDialogChild(item_revert, "CANVAS");
  char* filename = IupGetAttribute(canvas, "FILENAME");
  open_file(item_revert, filename);
  return IUP_DEFAULT;
}

int item_pagesetup_action_cb(Ihandle* item_pagesetup)
{
  Ihandle* canvas = IupGetDialogChild(item_pagesetup, "CANVAS");
  Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
  int margin_width = IupConfigGetVariableIntDef(config, "Print", "MarginWidth", 20)
  int margin_height = IupConfigGetVariableIntDef(config, "Print", "MarginHeight", 2

  if (IupGetParam("Page Setup", NULL, NULL, "Margin Width (mm): %i[1,]\nMargin Heig
  {
    IupConfigSetVariableInt(config, "Print", "MarginWidth", margin_width);
    IupConfigSetVariableInt(config, "Print", "MarginHeight", margin_height);
  }

  return IUP_DEFAULT;
}

int item_print_action_cb(Ihandle* item_print)
{
  Ihandle* canvas = IupGetDialogChild(item_print, "CANVAS");
  imImage* image;
  char* title = IupGetAttribute(IupGetDialog(item_print), "TITLE");

  cdCanvas* print_canvas = cdCreateCanvasf(CD_PRINTER, "%s -d", title);
  if (!print_canvas)
    return IUP_DEFAULT;

  /* do NOT draw the background, use the paper color */

  /* draw the image at the center of the canvas */
```

```
  image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  if (image)
  {
    int x, y, canvas_width, canvas_height, view_width, view_height;
    double canvas_width_mm, canvas_height_mm;
    Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
    int margin_width = IupConfigGetVariableIntDef(config, "Print", "MarginWidth", 2
    int margin_height = IupConfigGetVariableIntDef(config, "Print", "MarginHeight",

    cdCanvasGetSize(print_canvas, &canvas_width, &canvas_height, &canvas_width_mm, &

    /* convert to pixels */
    margin_width = (int)((margin_width * canvas_width) / canvas_width_mm);
    margin_height = (int)((margin_height * canvas_height) / canvas_height_mm);

    view_fit_rect(canvas_width - 2 * margin_width, canvas_height - 2 * margin_heigh
                  image->width, image->height,
                  &view_width, &view_height);

    x = (canvas_width - view_width) / 2;
    y = (canvas_height - view_height) / 2;

    cdCanvasPutImImage(print_canvas, image, x, y, view_width, view_height);
  }

  cdKillCanvas(print_canvas);
  return IUP_DEFAULT;
}

int item_exit_action_cb(Ihandle* item_exit)
{
  Ihandle* dlg = IupGetDialog(item_exit);
  Ihandle* config = (Ihandle*)IupGetAttribute(dlg, "CONFIG");
  Ihandle* canvas = IupGetDialogChild(dlg, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  Ihandle* toolbox = (Ihandle*)IupGetAttribute(dlg, "TOOLBOX");

  if (!save_check(item_exit))
    return IUP_IGNORE;  /* to abort the CLOSE_CB callback */

  if (IupGetInt(toolbox, "VISIBLE"))
  {
```

```
    IupConfigDialogClosed(config, toolbox, "Toolbox");
    IupHide(toolbox);
  }

  if (image)
    imImageDestroy(image);

  IupConfigDialogClosed(config, dlg, "MainWindow");
  IupConfigSave(config);
  IupDestroy(config);
  return IUP_CLOSE;
}

int item_copy_action_cb(Ihandle* item_copy)
{
  Ihandle* canvas = IupGetDialogChild(item_copy, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  Ihandle *clipboard = IupClipboard();
  IupSetAttribute(clipboard, "NATIVEIMAGE", (char*)IupGetImageNativeHandle(image));
  IupDestroy(clipboard);
  return IUP_DEFAULT;
}

int item_paste_action_cb(Ihandle* item_paste)
{
  if (save_check(item_paste))
  {
    Ihandle* canvas = IupGetDialogChild(item_paste, "CANVAS");

    Ihandle *clipboard = IupClipboard();
    imImage* image = IupGetNativeHandleImage(IupGetAttribute(clipboard, "NATIVEIMAG
    IupDestroy(clipboard);

    if (!image)
    {
      show_error("Invalid␣Clipboard␣Data", 1);
      return IUP_DEFAULT;
    }

    set_new_image(canvas, image, NULL, 1);  /* set dirty */
  }
  return IUP_DEFAULT;
```

```
}

int item_background_action_cb(Ihandle* item_background)
{
  Ihandle* canvas = IupGetDialogChild(item_background, "CANVAS");
  Ihandle* config = (Ihandle*)IupGetAttribute(canvas, "CONFIG");
  Ihandle* colordlg = IupColorDlg();
  const char* background = IupConfigGetVariableStrDef(config, "Canvas", "Background
  IupSetStrAttribute(colordlg, "VALUE", background);
  IupSetAttributeHandle(colordlg, "PARENTDIALOG", IupGetDialog(item_background));

  IupPopup(colordlg, IUP_CENTERPARENT, IUP_CENTERPARENT);

  if (IupGetInt(colordlg, "STATUS") == 1)
  {
    background = IupGetAttribute(colordlg, "VALUE");
    IupConfigSetVariableStr(config, "Canvas", "Background", background);

    IupUpdate(canvas);
  }

  IupDestroy(colordlg);
  return IUP_DEFAULT;
}

int item_zoomgrid_action_cb(Ihandle* ih)
{
  Ihandle* item_zoomgrid = IupGetDialogChild(ih, "ZOOMGRID");
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  Ihandle* config = (Ihandle*)IupGetAttribute(ih, "CONFIG");

  if (IupGetInt(item_zoomgrid, "VALUE"))
    IupSetAttribute(item_zoomgrid, "VALUE", "OFF");
  else
    IupSetAttribute(item_zoomgrid, "VALUE", "ON");

  IupConfigSetVariableStr(config, "Canvas", "ZoomGrid", IupGetAttribute(item_zoomgr

  IupUpdate(canvas);
  return IUP_DEFAULT;
}
```

```
int item_toolbar_action_cb(Ihandle* item_toolbar)
{
  Ihandle* toolbar = IupGetDialogChild(item_toolbar, "TOOLBAR");
  Ihandle* config = (Ihandle*)IupGetAttribute(item_toolbar, "CONFIG");

  toggle_bar_visibility(item_toolbar, toolbar);

  IupConfigSetVariableStr(config, "MainWindow", "Toolbar", IupGetAttribute(item_too
  return IUP_DEFAULT;
}

int item_toolbox_action_cb(Ihandle* item_toolbox)
{
  Ihandle* toolbox = (Ihandle*)IupGetAttribute(item_toolbox, "TOOLBOX");
  Ihandle* config = (Ihandle*)IupGetAttribute(item_toolbox, "CONFIG");

  if (IupGetInt(toolbox, "VISIBLE"))
  {
    IupSetAttribute(item_toolbox, "VALUE", "OFF");
    IupConfigDialogClosed(config, toolbox, "Toolbox");
    IupHide(toolbox);
  }
  else
  {
    IupSetAttribute(item_toolbox, "VALUE", "ON");
    IupConfigDialogShow(config, toolbox, "Toolbox");
  }

  IupConfigSetVariableStr(config, "MainWindow", "Toolbox", IupGetAttribute(item_too
  return IUP_DEFAULT;
}

int item_statusbar_action_cb(Ihandle* item_statusbar)
{
  Ihandle* statusbar = IupGetDialogChild(item_statusbar, "STATUSBAR");
  Ihandle* config = (Ihandle*)IupGetAttribute(item_statusbar, "CONFIG");

  toggle_bar_visibility(item_statusbar, statusbar);

  IupConfigSetVariableStr(config, "MainWindow", "Statusbar", IupGetAttribute(item_s
  return IUP_DEFAULT;
}
```

```
int item_help_action_cb(void)
{
  IupHelp("http://www.tecgraf.puc-rio.br/iup");
  return IUP_DEFAULT;
}

int item_about_action_cb(void)
{
  IupMessage("About", "␣␣␣Simple␣Paint\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣Scu
  return IUP_DEFAULT;
}

int toolbox_close_cb(Ihandle* toolbox)
{
  Ihandle* config = (Ihandle*)IupGetAttribute(toolbox, "CONFIG");
  Ihandle* canvas = (Ihandle*)IupGetAttribute(toolbox, "CANVAS");
  Ihandle* item_toolbox = IupGetDialogChild(canvas, "TOOLBOXMENU");

  IupConfigDialogClosed(config, toolbox, "Toolbox");

  IupSetAttribute(item_toolbox, "VALUE", "OFF");
  IupConfigSetVariableStr(config, "MainWindow", "Toolbox", "OFF");
  return IUP_DEFAULT;
}

int tool_action_cb(Ihandle* ih, int state)
{
  if (state == 1)
  {
    Ihandle* canvas = (Ihandle*)IupGetAttribute(ih, "CANVAS");
    int tool_index = IupGetInt(ih, "TOOLINDEX");
    IupSetInt(IupGetDialog(ih), "TOOLINDEX", tool_index);

    if (tool_index == 0)
      IupSetAttribute(canvas, "CURSOR", "ARROW");
    else
      IupSetAttribute(canvas, "CURSOR", "CROSS");

    if (tool_index == 8)
      tool_get_text(IupGetDialog(ih));
  }
```

```
  return IUP_DEFAULT;
}

int toolcolor_action_cb(Ihandle* ih)
{
  Ihandle* colordlg = IupColorDlg();
  const char* color = IupGetAttribute(ih, "BGCOLOR");
  IupSetStrAttribute(colordlg, "VALUE", color);
  IupSetAttributeHandle(colordlg, "PARENTDIALOG", IupGetDialog(ih));

  IupPopup(colordlg, IUP_CENTER, IUP_CENTER);

  if (IupGetInt(colordlg, "STATUS") == 1)
  {
    color = IupGetAttribute(colordlg, "VALUE");

    IupSetStrAttribute(ih, "BGCOLOR", color);
    IupSetStrAttribute(IupGetDialog(ih), "TOOLCOLOR", color);
  }

  IupDestroy(colordlg);
  return IUP_DEFAULT;
}

int toolwidth_valuechanged_cb(Ihandle* ih)
{
  char* value = IupGetAttribute(ih, "VALUE");
  IupSetStrAttribute(IupGetDialog(ih), "TOOLWIDTH", value);
  return IUP_DEFAULT;
}

int toolstyle_valuechanged_cb(Ihandle* ih)
{
  char* value = IupGetAttribute(ih, "VALUE");
  IupSetStrAttribute(IupGetDialog(ih), "TOOLSTYLE", value);
  return IUP_DEFAULT;
}

int toolfont_action_cb(Ihandle* ih)
{
  Ihandle* font_dlg = IupFontDlg();
  char* font = IupGetAttribute(ih, "TOOLFONT");
```

```
  IupSetAttributeHandle(font_dlg, "PARENTDIALOG", IupGetDialog(ih));
  IupSetStrAttribute(font_dlg, "VALUE", font);

  IupPopup(font_dlg, IUP_CENTER, IUP_CENTER);

  if (IupGetInt(font_dlg, "STATUS") == 1)
  {
    font = IupGetAttribute(font_dlg, "VALUE");
    IupSetStrAttribute(IupGetDialog(ih), "TOOLFONT", font);
  }
  IupDestroy(font_dlg);
  return IUP_DEFAULT;
}

int toolfilltol_valuechanged_cb(Ihandle* ih)
{
  Ihandle* filltol_label = IupGetDialogChild(ih, "FILLTOLLABEL");
  double value = IupGetDouble(ih, "VALUE");
  IupSetStrf(filltol_label, "TITLE", "Tol.: %.0f%%", value);
  IupSetDouble(IupGetDialog(ih), "TOOLFILLTOL", value);
  return IUP_DEFAULT;
}

int main_dlg_move_cb(Ihandle* dlg, int x, int y)
{
  Ihandle* toolbox = (Ihandle*)IupGetAttribute(dlg, "TOOLBOX");

  int old_x = IupGetInt(dlg, "_OLD_X");
  int old_y = IupGetInt(dlg, "_OLD_Y");

  if (old_x == x && old_y == y)
    return IUP_DEFAULT;

  if (IupGetInt(toolbox, "VISIBLE"))
  {
    int tb_x = IupGetInt(toolbox, "X");
    int tb_y = IupGetInt(toolbox, "Y");

    tb_x += x - old_x;
    tb_y += y - old_y;

    IupShowXY(toolbox, tb_x, tb_y);
```

```
  }

  IupSetInt(dlg, "_OLD_X", x);
  IupSetInt(dlg, "_OLD_Y", y);

  return IUP_DEFAULT;
}

int item_resize_action_cb(Ihandle* ih)
{
  Ihandle* config = (Ihandle*)IupGetAttribute(ih, "CONFIG");
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  imImage* new_image;
  int height = image->height,
      width = image->width;
  int quality = IupConfigGetVariableIntDef(config, "Image", "ResizeQuality", 1);
/* medium default */

  if (!IupGetParam("Resize", NULL, NULL,
                   "Width: %i [1,]\n"
                   "Height: %i [1,]\n"
                   "Quality: %l|low|medium|high|\n",
                   &width, &height, &quality, NULL))
    return IUP_DEFAULT;

  IupConfigSetVariableInt(config, "Image", "ResizeQuality", quality);

  new_image = imImageCreateBased(image, width, height, -1, -1);
  if (!new_image)
  {
    show_file_error(IM_ERR_MEM);
    return IUP_DEFAULT;
  }

  if (quality == 2)
    quality = 3; /* interpolation order can be 0, 1, and 3 */

  imProcessResize(image, new_image, quality);

  update_image(canvas, new_image, 1);    /* update size */
```

```
  return IUP_DEFAULT;
}

int item_mirror_action_cb(Ihandle* ih)
{
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  imImage* new_image = imImageClone(image);
  if (!new_image)
  {
    show_file_error(IM_ERR_MEM);
    return IUP_DEFAULT;
  }

  imProcessMirror(image, new_image);

  update_image(canvas, new_image, 0);

  return IUP_DEFAULT;
}

int item_flip_action_cb(Ihandle* ih)
{
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  imImage* new_image = imImageClone(image);
  if (!new_image)
  {
    show_file_error(IM_ERR_MEM);
    return IUP_DEFAULT;
  }

  imProcessFlip(image, new_image);

  update_image(canvas, new_image, 0);

  return IUP_DEFAULT;
}

int item_rotate180_action_cb(Ihandle* ih)
{
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
```

```
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  imImage* new_image = imImageClone(image);
  if (!new_image)
  {
    show_file_error(IM_ERR_MEM);
    return IUP_DEFAULT;
  }

  imProcessRotate180(image, new_image);

  update_image(canvas, new_image, 0);

  return IUP_DEFAULT;
}

int item_rotate90cw_action_cb(Ihandle* ih)
{
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  imImage* new_image = imImageCreateBased(image, image->height, image->width, -1, -
  if (!new_image)
  {
    show_file_error(IM_ERR_MEM);
    return IUP_DEFAULT;
  }

  imProcessRotate90(image, new_image, 1);

  update_image(canvas, new_image, 1);    /* update size */

  return IUP_DEFAULT;
}

int item_rotate90ccw_action_cb(Ihandle* ih)
{
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  imImage* new_image = imImageCreateBased(image, image->height, image->width, -1, -
  if (!new_image)
  {
    show_file_error(IM_ERR_MEM);
    return IUP_DEFAULT;
```

```
  }

  imProcessRotate90(image, new_image, -1);

  update_image(canvas, new_image, 1);    /* update size */

  return IUP_DEFAULT;
}

int item_negative_action_cb(Ihandle* ih)
{
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  imImage* new_image = imImageClone(image);
  if (!new_image)
  {
    show_file_error(IM_ERR_MEM);
    return IUP_DEFAULT;
  }

  imProcessNegative(image, new_image);

  update_image(canvas, new_image, 0);

  return IUP_DEFAULT;
}

static int brightcont_param_cb(Ihandle* dialog, int param_index, void* user_data)
{
  Ihandle* canvas = (Ihandle*)user_data;

  if (param_index == 0 || param_index == 1)
  {
    float param[2] = { 0, 0 };
    imImage* image = (imImage*)IupGetAttribute(canvas, "ORIGINAL_IMAGE");
    imImage* new_image = (imImage*)IupGetAttribute(canvas, "NEW_IMAGE");
    Ihandle* brightness_shift_param = (Ihandle*)IupGetAttribute(dialog, "PARAM0");
    Ihandle* contrast_factor_param = (Ihandle*)IupGetAttribute(dialog, "PARAM1");
    param[0] = IupGetFloat(brightness_shift_param, "VALUE");
    param[1] = IupGetFloat(contrast_factor_param, "VALUE");

    imProcessToneGamut(image, new_image, IM_GAMUT_BRIGHTCONT, param);
```

```
    IupSetAttribute(canvas, "IMAGE", (char*)new_image);
    IupUpdate(canvas);
  }
  else if (param_index != IUP_GETPARAM_INIT && param_index != IUP_GETPARAM_MAP)
  {
    /* restore original configuration */
    imImage* image = (imImage*)IupGetAttribute(canvas, "ORIGINAL_IMAGE");
    IupSetAttribute(canvas, "IMAGE", (char*)image);
    IupSetAttribute(canvas, "ORIGINAL_IMAGE", NULL);
    IupSetAttribute(canvas, "NEW_IMAGE", NULL);

    if (param_index == IUP_GETPARAM_BUTTON2)  /* cancel */
      IupUpdate(canvas);
  }

  return 1;
}

int item_brightcont_action_cb(Ihandle* ih)
{
  float param[2] = { 0, 0 };
  Ihandle* canvas = IupGetDialogChild(ih, "CANVAS");
  imImage* image = (imImage*)IupGetAttribute(canvas, "IMAGE");
  imImage* new_image = imImageClone(image);
  if (!new_image)
  {
    show_file_error(IM_ERR_MEM);
    return IUP_DEFAULT;
  }

  IupSetAttribute(canvas, "ORIGINAL_IMAGE", (char*)image);
  IupSetAttribute(canvas, "NEW_IMAGE", (char*)new_image);

  if (!IupGetParam("Brightness and Contrast", brightcont_param_cb, canvas,
                   "Brightness Shift: %r[-100,100]\n"
                   "Contrast Factor: %r[-100,100]\n",
                   &param[0], &param[1], NULL))
  {
    imImageDestroy(new_image);
          return IUP_DEFAULT;
  }
```

```
  imProcessToneGamut(image, new_image, IM_GAMUT_BRIGHTCONT, param);

  update_image(canvas, new_image, 0);

  return IUP_DEFAULT;
}


/********************************* Main *****************************************/


Ihandle* create_main_menu(Ihandle *config)
{
  Ihandle *menu, *sub_menu_file;
  Ihandle *file_menu, *item_exit, *item_new, *item_open, *item_save, *item_saveas,
  Ihandle *sub_menu_edit, *edit_menu, *item_copy, *item_paste, *item_print, *item_p
  Ihandle *sub_menu_help, *help_menu, *item_help, *item_about;
  Ihandle *sub_menu_view, *view_menu, *item_toolbar, *item_statusbar;
  Ihandle *item_zoomin, *item_zoomout, *item_actualsize;
  Ihandle *recent_menu, *item_background, *item_toolbox, *item_zoomgrid;
  Ihandle *sub_menu_image, *image_menu;

  item_new = IupItem("&New\tCtrl+N", NULL);
  IupSetAttribute(item_new, "IMAGE", "IUP_FileNew");
  IupSetCallback(item_new, "ACTION", (Icallback)item_new_action_cb);

  item_open = IupItem("&Open...\tCtrl+O", NULL);
  IupSetAttribute(item_open, "IMAGE", "IUP_FileOpen");
  IupSetCallback(item_open, "ACTION", (Icallback)item_open_action_cb);

  item_save = IupItem("&Save\tCtrl+S", NULL);
  IupSetAttribute(item_save, "NAME", "ITEM_SAVE");
  IupSetAttribute(item_save, "IMAGE", "IUP_FileSave");
  IupSetCallback(item_save, "ACTION", (Icallback)item_save_action_cb);

  item_saveas = IupItem("Save␣&As...", NULL);
  IupSetAttribute(item_saveas, "NAME", "ITEM_SAVEAS");
  IupSetCallback(item_saveas, "ACTION", (Icallback)item_saveas_action_cb);

  item_revert = IupItem("&Revert", NULL);
  IupSetAttribute(item_revert, "NAME", "ITEM_REVERT");
```

```
IupSetCallback(item_revert, "ACTION", (Icallback)item_revert_action_cb);

item_pagesetup = IupItem("Page Set&up...", NULL);
IupSetCallback(item_pagesetup, "ACTION", (Icallback)item_pagesetup_action_cb);

item_print = IupItem("&Print...\tCtrl+P", NULL);
IupSetCallback(item_print, "ACTION", (Icallback)item_print_action_cb);

item_exit = IupItem("E&xit", NULL);
IupSetCallback(item_exit, "ACTION", (Icallback)item_exit_action_cb);

item_copy = IupItem("&Copy\tCtrl+C", NULL);
IupSetAttribute(item_copy, "NAME", "ITEM_COPY");
IupSetAttribute(item_copy, "IMAGE", "IUP_EditCopy");
IupSetCallback(item_copy, "ACTION", (Icallback)item_copy_action_cb);

item_paste = IupItem("&Paste\tCtrl+V", NULL);
IupSetAttribute(item_paste, "NAME", "ITEM_PASTE");
IupSetAttribute(item_paste, "IMAGE", "IUP_EditPaste");
IupSetCallback(item_paste, "ACTION", (Icallback)item_paste_action_cb);

item_zoomin = IupItem("Zoom &In\tCtrl++", NULL);
IupSetAttribute(item_zoomin, "IMAGE", "IUP_ZoomIn");
IupSetCallback(item_zoomin, "ACTION", (Icallback)zoomin_action_cb);

item_zoomout = IupItem("Zoom &Out\tCtrl+-", NULL);
IupSetAttribute(item_zoomout, "IMAGE", "IUP_ZoomOut");
IupSetCallback(item_zoomout, "ACTION", (Icallback)zoomout_action_cb);

item_actualsize = IupItem("&Actual Size\tCtrl+0", NULL);
IupSetAttribute(item_actualsize, "IMAGE", "IUP_ZoomActualSize");
IupSetCallback(item_actualsize, "ACTION", (Icallback)actualsize_action_cb);

item_zoomgrid = IupItem("&Zoom Grid", NULL);
IupSetCallback(item_zoomgrid, "ACTION", (Icallback)item_zoomgrid_action_cb);
IupSetAttribute(item_zoomgrid, "NAME", "ZOOMGRID");
IupSetAttribute(item_zoomgrid, "VALUE", "ON");  /* default is ON */

item_background = IupItem("&Background...", NULL);
IupSetCallback(item_background, "ACTION", (Icallback)item_background_action_cb);

item_toolbar = IupItem("&Toobar", NULL);
```

```
  IupSetCallback(item_toolbar, "ACTION", (Icallback)item_toolbar_action_cb);
  IupSetAttribute(item_toolbar, "VALUE", "ON");    /* default is ON */

  item_toolbox = IupItem("&Toobox", NULL);
  IupSetCallback(item_toolbox, "ACTION", (Icallback)item_toolbox_action_cb);
  IupSetAttribute(item_toolbox, "NAME", "TOOLBOXMENU");
  IupSetAttribute(item_toolbox, "VALUE", "ON");    /* default is ON */

  item_statusbar = IupItem("&Statusbar", NULL);
  IupSetCallback(item_statusbar, "ACTION", (Icallback)item_statusbar_action_cb);
  IupSetAttribute(item_statusbar, "VALUE", "ON");  /* default is ON */

  item_help = IupItem("&Help...", NULL);
  IupSetCallback(item_help, "ACTION", (Icallback)item_help_action_cb);

  item_about = IupItem("&About...", NULL);
  IupSetCallback(item_about, "ACTION", (Icallback)item_about_action_cb);

  recent_menu = IupMenu(NULL);

  file_menu = IupMenu(
    item_new,
    item_open,
    item_save,
    item_saveas,
    item_revert,
    IupSeparator(),
    item_pagesetup,
    item_print,
    IupSeparator(),
    IupSubmenu("Recent␣&Files", recent_menu),
    IupSeparator(),
    item_exit,
    NULL);
  edit_menu = IupMenu(
    item_copy,
    item_paste,
    NULL);
  view_menu = IupMenu(
    item_zoomin,
    item_zoomout,
    item_actualsize,
```

```
    item_zoomgrid,
    IupSeparator(),
    item_background,
    IupSeparator(),
    item_toolbar,
    item_toolbox,
    item_statusbar,
    NULL);
  image_menu = IupMenu(
    IupSetCallbacks(IupItem("&Resize...", NULL), "ACTION", item_resize_action_cb, N
    IupSetCallbacks(IupItem("&Mirror", NULL), "ACTION", item_mirror_action_cb, NULL
    IupSetCallbacks(IupItem("&Flip", NULL), "ACTION", item_flip_action_cb, NULL),
    IupSetCallbacks(IupItem("&Rotate 180º", NULL), "ACTION", item_rotate180_action_
    IupSetCallbacks(IupItem("&Rotate +90º (clock-wise)", NULL), "ACTION", item_rota
    IupSetCallbacks(IupItem("&Rotate -90º (counter-clock)", NULL), "ACTION", item_r
    IupSeparator(),
    IupSetCallbacks(IupItem("&Negative", NULL), "ACTION", item_negative_action_cb,
    IupSetCallbacks(IupItem("&Brightness and Contrast...", NULL), "ACTION", item_br
    NULL);
  help_menu = IupMenu(
    item_help,
    item_about,
    NULL);

  IupSetCallback(file_menu, "OPEN_CB", (Icallback)file_menu_open_cb);
  IupSetCallback(edit_menu, "OPEN_CB", (Icallback)edit_menu_open_cb);

  sub_menu_file = IupSubmenu("&File", file_menu);
  sub_menu_edit = IupSubmenu("&Edit", edit_menu);
  sub_menu_view = IupSubmenu("&View", view_menu);
  sub_menu_image = IupSubmenu("&Image", image_menu);
  sub_menu_help = IupSubmenu("&Help", help_menu);

  menu = IupMenu(
    sub_menu_file,
    sub_menu_edit,
    sub_menu_view,
    sub_menu_image,
    sub_menu_help,
    NULL);

  /* Initialize variables from the configuration file */
```

```
  IupConfigRecentInit(config, recent_menu, config_recent_cb, 10);

  if (!IupConfigGetVariableIntDef(config, "Canvas", "ZoomGrid", 1))
    IupSetAttribute(item_zoomgrid, "VALUE", "OFF");

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Toolbar", 1))
    IupSetAttribute(item_toolbar, "VALUE", "OFF");

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Toolbox", 1))
    IupSetAttribute(item_toolbox, "VALUE", "OFF");

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Statusbar", 1))
    IupSetAttribute(item_statusbar, "VALUE", "OFF");

  return menu;
}

Ihandle* create_toolbar(Ihandle *config)
{
  Ihandle *toolbar;
  Ihandle *btn_copy, *btn_paste, *btn_new, *btn_open, *btn_save, *btn_zoomgrid;

  IupSetHandle("PaintZoomGrid", load_image_PaintZoomGrid());

  btn_new = IupButton(NULL, NULL);
  IupSetAttribute(btn_new, "IMAGE", "IUP_FileNew");
  IupSetAttribute(btn_new, "FLAT", "Yes");
  IupSetCallback(btn_new, "ACTION", (Icallback)item_new_action_cb);
  IupSetAttribute(btn_new, "TIP", "New (Ctrl+N)");
  IupSetAttribute(btn_new, "CANFOCUS", "No");

  btn_open = IupButton(NULL, NULL);
  IupSetAttribute(btn_open, "IMAGE", "IUP_FileOpen");
  IupSetAttribute(btn_open, "FLAT", "Yes");
  IupSetCallback(btn_open, "ACTION", (Icallback)item_open_action_cb);
  IupSetAttribute(btn_open, "TIP", "Open (Ctrl+O)");
  IupSetAttribute(btn_open, "CANFOCUS", "No");

  btn_save = IupButton(NULL, NULL);
  IupSetAttribute(btn_save, "IMAGE", "IUP_FileSave");
  IupSetAttribute(btn_save, "FLAT", "Yes");
```

```
  IupSetCallback(btn_save, "ACTION", (Icallback)item_save_action_cb);
  IupSetAttribute(btn_save, "TIP", "Save (Ctrl+S)");
  IupSetAttribute(btn_save, "CANFOCUS", "No");

  btn_copy = IupButton(NULL, NULL);
  IupSetAttribute(btn_copy, "IMAGE", "IUP_EditCopy");
  IupSetAttribute(btn_copy, "FLAT", "Yes");
  IupSetCallback(btn_copy, "ACTION", (Icallback)item_copy_action_cb);
  IupSetAttribute(btn_copy, "TIP", "Copy (Ctrl+C)");
  IupSetAttribute(btn_copy, "CANFOCUS", "No");

  btn_paste = IupButton(NULL, NULL);
  IupSetAttribute(btn_paste, "IMAGE", "IUP_EditPaste");
  IupSetAttribute(btn_paste, "FLAT", "Yes");
  IupSetCallback(btn_paste, "ACTION", (Icallback)item_paste_action_cb);
  IupSetAttribute(btn_paste, "TIP", "Paste (Ctrl+V)");
  IupSetAttribute(btn_paste, "CANFOCUS", "No");

  btn_zoomgrid = IupButton(NULL, NULL);
  IupSetAttribute(btn_zoomgrid, "IMAGE", "PaintZoomGrid");
  IupSetAttribute(btn_zoomgrid, "FLAT", "Yes");
  IupSetCallback(btn_zoomgrid, "ACTION", (Icallback)item_zoomgrid_action_cb);
  IupSetAttribute(btn_zoomgrid, "TIP", "Zoom Grid");
  IupSetAttribute(btn_paste, "CANFOCUS", "No");

  toolbar = IupHbox(
    btn_new,
    btn_open,
    btn_save,
    IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
    btn_copy,
    btn_paste,
    IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
    btn_zoomgrid,
    NULL);
  IupSetAttribute(toolbar, "MARGIN", "5x5");
  IupSetAttribute(toolbar, "GAP", "2");
  IupSetAttribute(toolbar, "NAME", "TOOLBAR");

  /* Initialize variables from the configuration file */

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Toolbar", 1))
```

```
  {
    IupSetAttribute(toolbar, "FLOATING", "YES");
    IupSetAttribute(toolbar, "VISIBLE", "NO");
  }

  return toolbar;
}

void create_toolbox(Ihandle* parent_dlg, Ihandle *config)
{
  Ihandle *toolbox, *gbox, *vbox;
  Ihandle* canvas = IupGetDialogChild(parent_dlg, "CANVAS");

  IupSetHandle("PaintPointer", load_image_Pointer());
  IupSetHandle("PaintColorPicker", load_image_PaintColorPicker());
  IupSetHandle("PaintPencil", load_image_PaintPencil());
  IupSetHandle("PaintLine", load_image_PaintLine());
  IupSetHandle("PaintEllipse", load_image_PaintEllipse());
  IupSetHandle("PaintRect", load_image_PaintRect());
  IupSetHandle("PaintOval", load_image_PaintOval());
  IupSetHandle("PaintBox", load_image_PaintBox());
  IupSetHandle("PaintFill", load_image_PaintFill());
  IupSetHandle("PaintText", load_image_PaintText());

  gbox = IupGridBox(
    IupSetCallbacks(IupSetAttributes(IupToggle(NULL, NULL), "TOOLINDEX=0,␣IMAGE=Pai
    IupSetCallbacks(IupSetAttributes(IupToggle(NULL, NULL), "TOOLINDEX=1,␣IMAGE=Pai
    IupSetCallbacks(IupSetAttributes(IupToggle(NULL, NULL), "TOOLINDEX=2,␣IMAGE=Pai
    IupSetCallbacks(IupSetAttributes(IupToggle(NULL, NULL), "TOOLINDEX=3,␣IMAGE=Pai
    IupSetCallbacks(IupSetAttributes(IupToggle(NULL, NULL), "TOOLINDEX=4,␣IMAGE=Pai
    IupSetCallbacks(IupSetAttributes(IupToggle(NULL, NULL), "TOOLINDEX=5,␣IMAGE=Pai
    IupSetCallbacks(IupSetAttributes(IupToggle(NULL, NULL), "TOOLINDEX=6,␣IMAGE=Pai
    IupSetCallbacks(IupSetAttributes(IupToggle(NULL, NULL), "TOOLINDEX=7,␣IMAGE=Pai
    IupSetCallbacks(IupSetAttributes(IupToggle(NULL, NULL), "TOOLINDEX=8,␣IMAGE=Pai
    IupSetCallbacks(IupSetAttributes(IupToggle(NULL, NULL), "TOOLINDEX=9,␣IMAGE=Pai
    NULL);
  IupSetAttribute(gbox, "GAPCOL", "2");
  IupSetAttribute(gbox, "GAPLIN", "2");
  IupSetAttribute(gbox, "MARGIN", "5x10");
  IupSetAttribute(gbox, "NUMDIV", "2");

  vbox = IupVbox(
```

```
    IupRadio(gbox),
    IupFrame(IupSetAttributes(IupVbox(
      IupSetAttributes(IupLabel("Color:"), "EXPAND=HORIZONTAL"),
      IupSetCallbacks(IupSetAttributes(IupButton(NULL, NULL), "NAME=COLOR, BGCOLOR=
      IupSetAttributes(IupLabel("Width:"), "EXPAND=HORIZONTAL"),
      IupSetCallbacks(IupSetAttributes(IupText(NULL), "SPIN=Yes, SPINMIN=1, RASTERS
      IupSetAttributes(IupLabel("Style:"), "EXPAND=HORIZONTAL"),
      IupSetCallbacks(IupSetAttributes(IupList(NULL), "DROPDOWN=Yes, VALUE=1, 1=\"_
      IupSetAttributes(IupLabel("Tol.: 50%"), "EXPAND=HORIZONTAL, NAME=FILLTOLLABEL
      IupSetCallbacks(IupSetAttributes(IupVal(NULL), "NAME=FILLTOL, RASTERSIZE=60x3
      IupSetAttributes(IupLabel("Font:"), "EXPAND=HORIZONTAL"),
      IupSetCallbacks(IupSetAttributes(IupButton("F", NULL), "NAME=FONT, RASTERSIZE
      NULL), "MARGIN=3x2, GAP=2, ALIGNMENT=ACENTER")),
    NULL);
  IupSetAttribute(vbox, "NMARGIN", "2x2");
  IupSetAttribute(vbox, "ALIGNMENT", "ACENTER");

  toolbox = IupDialog(vbox);
  IupSetAttribute(toolbox, "DIALOGFRAME", "Yes");
  IupSetAttribute(toolbox, "TITLE", "Tools");
  IupSetAttribute(toolbox, "FONTSIZE", "8");
  IupSetAttribute(toolbox, "TOOLBOX", "Yes");
  IupSetCallback(toolbox, "CLOSE_CB", (Icallback)toolbox_close_cb);
  IupSetAttributeHandle(toolbox, "PARENTDIALOG", parent_dlg);

  IupSetAttribute(toolbox, "TOOLCOLOR", "0 0 0");
  IupSetAttribute(toolbox, "TOOLWIDTH", "1");
  IupSetAttribute(toolbox, "TOOLSTYLE", "1");
  IupSetAttribute(toolbox, "TOOLFILLTOL", "50");
  IupSetStrAttribute(toolbox, "TOOLFONT", IupGetAttribute(parent_dlg, "FONT"));

  IupSetAttribute(toolbox, "CONFIG", (char*)config);
  IupSetAttribute(toolbox, "CANVAS", (char*)canvas);

  IupSetAttribute(parent_dlg, "TOOLBOX", (char*)toolbox);

  /* Initialize variables from the configuration file */

  if (IupConfigGetVariableIntDef(config, "MainWindow", "Toolbox", 1))
  {
    /* configure the very first time to be aligned with the main window */
    if (!IupConfigGetVariableStr(config, "Toolbox", "X"))
```

```
    {
      Ihandle* canvas = IupGetDialogChild(parent_dlg, "CANVAS");
      int x = IupGetInt(canvas, "X");
      int y = IupGetInt(canvas, "Y");
      IupConfigSetVariableInt(config, "Toolbox", "X", x);
      IupConfigSetVariableInt(config, "Toolbox", "Y", y);
    }

    IupConfigDialogShow(config, toolbox, "Toolbox");
  }
}

Ihandle* create_statusbar(Ihandle *config)
{
  Ihandle *statusbar;

  statusbar = IupHbox(
    IupSetAttributes(IupLabel("(0, 0) = 0 0 0"), "EXPAND=HORIZONTAL, PADDING=10
    IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
    IupSetAttributes(IupLabel("0 x 0"), "SIZE=70x, PADDING=10x5, NAME=SIZELABEL, AL
    IupSetAttributes(IupLabel(NULL), "SEPARATOR=VERTICAL"),
    IupSetAttributes(IupLabel("100%"), "SIZE=30x, PADDING=10x5, NAME=ZOOMLABEL, ALIG
    IupSetCallbacks(IupSetAttributes(IupButton(NULL, NULL), "IMAGE=IUP_ZoomOut, FLA
    IupSetCallbacks(IupSetAttributes(IupVal(NULL), "VALUE=0, MIN=-6, MAX=6, RASTERS
    IupSetCallbacks(IupSetAttributes(IupButton(NULL, NULL), "IMAGE=IUP_ZoomIn, FLAT
    IupSetCallbacks(IupSetAttributes(IupButton(NULL, NULL), "IMAGE=IUP_ZoomActualSi
    NULL);
  IupSetAttribute(statusbar, "NAME", "STATUSBAR");
  IupSetAttribute(statusbar, "ALIGNMENT", "ACENTER");

  /* Initialize variables from the configuration file */

  if (!IupConfigGetVariableIntDef(config, "MainWindow", "Statusbar", 1))
  {
    IupSetAttribute(statusbar, "FLOATING", "YES");
    IupSetAttribute(statusbar, "VISIBLE", "NO");
  }

  return statusbar;
}

Ihandle* create_main_dialog(Ihandle *config)
```

```
{
  Ihandle *dlg, *vbox, *canvas;

#ifdef USE_OPENGL
  canvas = IupGLCanvas(NULL);
  IupSetAttribute(canvas, "BUFFER", "DOUBLE");
#else
  canvas = IupCanvas(NULL);
#endif
  IupSetAttribute(canvas, "NAME", "CANVAS");
  IupSetAttribute(canvas, "SCROLLBAR", "Yes");
  IupSetAttribute(canvas, "DIRTY", "NO");  /* custom attribute */
  IupSetCallback(canvas, "ACTION", (Icallback)canvas_action_cb);
  IupSetCallback(canvas, "DROPFILES_CB", (Icallback)dropfiles_cb);
  IupSetCallback(canvas, "MAP_CB", (Icallback)canvas_map_cb);
  IupSetCallback(canvas, "UNMAP_CB", (Icallback)canvas_unmap_cb);
  IupSetCallback(canvas, "WHEEL_CB", (Icallback)canvas_wheel_cb);
  IupSetCallback(canvas, "RESIZE_CB", (Icallback)canvas_resize_cb);
  IupSetCallback(canvas, "MOTION_CB", (Icallback)canvas_motion_cb);
  IupSetCallback(canvas, "BUTTON_CB", (Icallback)canvas_button_cb);

  vbox = IupVbox(
    create_toolbar(config),
    canvas,
    create_statusbar(config),
    NULL);

  dlg = IupDialog(vbox);
  IupSetAttributeHandle(dlg, "MENU", create_main_menu(config));
  IupSetAttribute(dlg, "SIZE", "HALFxHALF");
  IupSetCallback(dlg, "CLOSE_CB", (Icallback)item_exit_action_cb);
  IupSetCallback(dlg, "DROPFILES_CB", (Icallback)dropfiles_cb);
  IupSetCallback(dlg, "MOVE_CB", (Icallback)main_dlg_move_cb);

  IupSetCallback(dlg, "K_cN", (Icallback)item_new_action_cb);
  IupSetCallback(dlg, "K_cO", (Icallback)item_open_action_cb);
  IupSetCallback(dlg, "K_cS", (Icallback)item_save_action_cb);
  IupSetCallback(dlg, "K_cV", (Icallback)item_paste_action_cb);
  IupSetCallback(dlg, "K_cC", (Icallback)item_copy_action_cb);
  IupSetCallback(dlg, "K_cP", (Icallback)item_print_action_cb);
  IupSetCallback(dlg, "K_cMinus", (Icallback)zoomout_action_cb);
  IupSetCallback(dlg, "K_cPlus", (Icallback)zoomin_action_cb);
```

```
  IupSetCallback(dlg, "K_cEqual", (Icallback)zoomin_action_cb);
  IupSetCallback(dlg, "K_c0", (Icallback)actualsize_action_cb);

  /* parent for pre-defined dialogs in closed functions (IupMessage and IupAlarm) *
  IupSetAttributeHandle(NULL, "PARENTDIALOG", dlg);

  IupSetAttribute(dlg, "CONFIG", (char*)config);

  return dlg;
}

int main(int argc, char **argv)
{
  Ihandle *dlg, *config;

  IupOpen(&argc, &argv);
  IupImageLibOpen();
#ifdef USE_OPENGL
  IupGLCanvasOpen();
#endif
#ifdef USE_CONTEXTPLUS
  cdInitContextPlus();
#endif

  config = IupConfig();
  IupSetAttribute(config, "APP_NAME", "simple_paint");
  IupConfigLoad(config);

  dlg = create_main_dialog(config);

  /* show the dialog at the last position, with the last size */
  IupConfigDialogShow(config, dlg, "MainWindow");

  /* create and show the toolbox */
  create_toolbox(dlg, config);

  /* open a file from the command line (allow file association in Windows) */
  if (argc > 1 && argv[1])
  {
    const char* filename = argv[1];
    open_file(dlg, filename);
  }
```

```
  /* initialize the current file, if not already loaded */
  check_new_file(dlg);

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}

/* Possible Enhancements:
   - Save last used toolbox options in configuration file?
   - Hide/show toolbox options according to selected tool
   - Capture image from Camera using IM
   - Undo/Redo
   - Secondary color for drawing of shapes with both outline and filled at the same
   - Alpha for colors
   - Area Selection
*/
```

Lua语言

```
require("imlua")
require("imlua_process")
require("iuplua")
require("iupluaimglib")
require("iupluaim")
require("cdlua")
require("iupluacd")
require("cdluaim")


--********************************** Images ************************************


function load_image_PaintLine()
  local PaintLine = iup.imagergba
  {
    width = 16,
    height = 16,
    pixels = {
      0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 255, 255, 255, 0, 255, 255, 2
      0, 0, 0, 2, 0, 0, 0, 5, 0, 0, 0, 12, 0, 0, 0, 0, 255, 255, 255, 0, 255, 255,
```

```
      0, 0, 0, 0, 0, 0, 0, 13, 0, 0, 0, 239, 0, 0, 0, 60, 255, 255, 255, 0, 255, 25
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 84, 0, 0, 0, 239, 0, 0, 0, 60, 255, 255, 255
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 84, 0, 0, 0, 2
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
    },
  }
  return PaintLine
end

function load_image_PaintPointer()
  local PaintPointer = iup.imagergba
  {
    width = 16,
    height = 16,
    pixels = {
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 162, 180, 203, 255, 162
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 162, 180, 203, 255, 162
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 162, 180, 203, 255, 240
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 162, 180, 203, 255, 255
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 158, 176, 200, 255, 255
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 153, 172, 195, 255, 255
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 148, 166, 189, 255, 255
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 141, 159, 183, 255, 254
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 134, 152, 176, 255, 249
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 127, 145, 170, 255, 241
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 119, 138, 163, 255, 203
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 112, 131, 156, 255, 82,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 105, 124, 150, 255, 255
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
    },
```

```
  }
  return PaintPointer
end

function load_image_PaintPencil()
  local PaintPencil = iup.imagergba
  {
    width = 16,
    height = 16,
    pixels = {
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 175,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 175, 158, 120, 66, 179,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 152, 157, 163, 255, 199
      255, 255, 255, 0, 255, 255, 255, 0, 152, 157, 163, 118, 152, 157, 163, 255, 2
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 118, 110, 113, 117, 255, 81, 84,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 224, 0, 0, 0, 118, 81, 84, 87, 1
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
    },
  }
  return PaintPencil
end

function load_image_PaintColorPicker()
  local PaintColorPicker = iup.imagergba
  {
    width = 16,
    height = 16,
    pixels = {
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
```

```
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 152,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 152,
      255, 255, 255, 0, 43, 134, 240, 46, 43, 134, 240, 107, 152, 166, 185, 255, 25
      255, 255, 255, 0, 43, 133, 239, 170, 43, 133, 240, 255, 87, 105, 130, 255, 79
      255, 255, 255, 0, 42, 120, 237, 46, 42, 124, 238, 107, 42, 125, 238, 161, 42,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
    },
  }
  return PaintColorPicker
end

function load_image_PaintEllipse()
  local PaintEllipse = iup.imagergba
  {
    width = 16,
    height = 16,
    pixels = {
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 16, 0, 0, 0, 1
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 24, 0, 0, 0, 215, 0, 0, 0, 143,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 199, 0, 0, 0, 120, 255, 255, 255
      255, 255, 255, 0, 0, 0, 0, 60, 0, 0, 0, 211, 255, 255, 255, 0, 255, 255, 255,
      255, 255, 255, 0, 0, 0, 0, 120, 0, 0, 0, 139, 255, 255, 255, 0, 255, 255, 255
      255, 255, 255, 0, 0, 0, 0, 120, 0, 0, 0, 135, 255, 255, 255, 0, 255, 255, 255
      255, 255, 255, 0, 0, 0, 0, 68, 0, 0, 0, 211, 255, 255, 255, 0, 255, 255, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 199, 0, 0, 0, 120, 255, 255, 255
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 36, 0, 0, 0, 231, 0, 0, 0, 135,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 20, 0, 0, 0, 1
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
    },
  }
  return PaintEllipse
end
```

```
function load_image_PaintRect()
  local PaintRect = iup.imagergba
  {
    width = 16,
    height = 16,
    pixels = {
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 254, 0, 0, 0, 247, 0, 0, 0, 239,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 0, 255, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
    },
  }
  return PaintRect
end

function load_image_PaintOval()
  local PaintOval = iup.imagergba
  {
    width = 16,
    height = 16,
    pixels = {
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 80, 0, 0, 0, 2
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 48, 0, 0, 0, 251, 0, 0, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 183, 0, 0, 0, 255, 0, 0, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 247, 0, 0, 0, 255, 0, 0, 0, 255,
```

```
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 247, 0, 0, 0, 255, 0, 0, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 183, 0, 0, 0, 255, 0, 0, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 56, 0, 0, 0, 251, 0, 0, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 84, 0, 0, 0, 2
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
    },
  }
  return PaintOval
end

function load_image_PaintBox()
  local PaintBox = iup.imagergba
  {
    width = 16,
    height = 16,
    pixels = {
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 254, 0, 0, 0, 247, 0, 0, 0, 239,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
    },
  }
  return PaintBox
end

function load_image_PaintZoomGrid()
  local PaintZoomGrid = iup.imagergba
```

```
  {
    width = 16,
    height = 16,
    pixels = {
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 110, 155, 223, 255, 255
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 110, 155, 223, 255, 255
      255, 255, 255, 0, 110, 155, 223, 255, 110, 155, 223, 255, 107, 152, 222, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 106, 149, 219, 255, 255
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 103, 148, 216, 255, 255
      255, 255, 255, 0, 104, 151, 219, 255, 104, 148, 216, 255, 100, 145, 213, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 97, 142, 210, 255, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 93, 138, 206, 255, 255,
      255, 255, 255, 0, 95, 140, 212, 255, 92, 137, 207, 255, 88, 133, 203, 255, 84
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 84, 131, 199, 255, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 82, 125, 195, 255, 255,
      255, 255, 255, 0, 84, 129, 199, 255, 80, 125, 195, 255, 76, 121, 191, 255, 72
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 72, 117, 187, 255, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 68, 113, 185, 255, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
    },
  }
  return PaintZoomGrid
end

function load_image_PaintFill()
  local PaintFill = iup.imagergba
  {
    width = 16,
    height = 16,
    pixels = {
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 50, 53, 53, 86, 5
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 52, 54, 54, 122,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 77, 77, 76, 154,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 124, 126, 129, 19
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 130, 141, 158, 45, 130, 143,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 127, 148, 170, 24, 134, 155, 174, 172, 21
      0, 0, 0, 0, 0, 0, 0, 0, 134, 147, 174, 19, 122, 146, 168, 179, 216, 223, 228,
      0, 0, 0, 0, 120, 134, 161, 19, 126, 147, 168, 157, 209, 217, 225, 245, 217, 2
      122, 132, 160, 27, 102, 124, 149, 172, 194, 209, 220, 241, 213, 228, 239, 246
      125, 134, 152, 55, 111, 131, 154, 211, 189, 201, 214, 255, 214, 223, 234, 254
      0, 255, 255, 1, 134, 140, 167, 38, 137, 155, 179, 206, 218, 223, 230, 254, 24
```

```
      0, 0, 0, 0, 0, 0, 255, 1, 157, 173, 195, 47, 148, 166, 191, 205, 204, 210, 22
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 127, 142, 165, 34, 103, 124, 151, 197, 15
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 102, 110, 119, 30, 73, 96, 12
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 49, 82, 115, 31,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 109,
    },
  }
  return PaintFill
end

function load_image_PaintText()
  local PaintText = iup.imagergba
  {
    width = 16,
    height = 16,
    pixels = {
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 36, 0, 0, 0, 96, 0, 0, 0, 96, 0,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 96, 0, 0, 0, 255, 0, 0, 0, 203,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 96, 0, 0, 0, 163, 255, 255, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, 96, 0, 0, 0, 48, 255, 255, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
      255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255,
    },
  }
  return PaintText
end


--******************************** Utilities ***********************************

function str_splitfilename(filename)
  return string.match(filename, "(.-)([^\\/]-%.?([^%.\\/]*))$")
```

```
end

function str_fileext(filename)
  local path, title, ext = str_splitfilename(filename)
  return ext
end

function str_filetitle(filename)
  local path, title, ext = str_splitfilename(filename)
  return title
end

function show_error(message, is_error)
  local dlg = iup.messagedlg{
    parentdialog = iup.GetGlobal("PARENTDIALOG"),
    buttons = "OK",
    value = message,
  }
  if (is_error) then
    dlg.dialogtype = "ERROR"
    dlg.title = "Error"
  else
    dlg.dialogtype = "WARNING"
    dlg.title = "Warning"
  end
  dlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)
  dlg:destroy()
end

function show_file_error(err)
  show_error(im.ErrorStr(err), true)
end

function read_file(filename)
  local image, err = im.FileImageLoadBitmap(filename, 0)
  if (err) then
    show_file_error(err)
  end
  return image
end

function write_file(filename, image)
```

```
  local format = image:GetAttribString("FileFormat")
  local err = im.FileImageSave(filename, format, image)
  if (err and err ~= im.ERR_NONE) then
    show_file_error(err)
    return false
  end
  return true
end

-- extracted from the SCROLLBAR attribute documentation
function scroll_update(ih, view_width, view_height)
  -- view_width and view_height is the virtual space size
  -- here we assume XMIN=0, XMAX=1, YMIN=0, YMAX=1
  local scrollbar_size = tonumber(iup.GetGlobal("SCROLLBARSIZE"))
  local border = 1
  if (ih.border ~= "YES") then
    border = 0
  end

  local elem_width, elem_height = string.match(ih.rastersize, "(%d*)x(%d*)")
  elem_width = tonumber(elem_width)
  elem_height = tonumber(elem_height)

  -- if view is greater than canvas in one direction,
  -- then it has scrollbars,
  -- but this affects the opposite direction
  elem_width = elem_width - 2 * border  -- remove BORDER (always size=1)
  elem_height = elem_height - 2 * border
  local canvas_width = elem_width
  local canvas_height = elem_height
  if (view_width > elem_width) then  -- check for horizontal scrollbar
    canvas_height = canvas_height - scrollbar_size  -- affect vertical size
  end
  if (view_height > elem_height) then
    canvas_width = canvas_width - scrollbar_size
  end
  if (view_width <= elem_width and view_width > canvas_width) then  -- check if sti
    canvas_height = canvas_height - scrollbar_size
  end
  if (view_height <= elem_height and view_height > canvas_height) then
    canvas_width = canvas_width - scrollbar_size
  end
```

```
  if (canvas_width < 0) then canvas_width = 0 end
  if (canvas_height < 0) then canvas_height = 0 end

  ih.dx = canvas_width / view_width
  ih.dy = canvas_height / view_height
end

function scroll_calc_center(ih)
  local x = tonumber(ih.posx) + tonumber(ih.dx) / 2
  local y = tonumber(ih.posy) + tonumber(ih.dy) / 2
  return x, y
end

function scroll_center(ih, old_center_x, old_center_y)
  -- always update the scroll position
  -- keeping it proportional to the old position
  -- relative to the center of the ih.

  local dx = tonumber(ih.dx)
  local dy = tonumber(ih.dy)

  local posx = old_center_x - dx / 2
  local posy = old_center_y - dy / 2

  if (posx < 0) then posx = 0 end
  if (posx > 1 - dx) then posx = 1 - dx end

  if (posy < 0) then posy = 0 end
  if (posy > 1 - dy) then posy = 1 - dy end

  ih.posx = posx
  ih.posy = posy
end

function scroll_move(ih, canvas_width, canvas_height, move_x, move_y, view_width, v
  local posy = 0
  local posx = 0

  if (move_x == 0 and move_y == 0) then
    return
  end
  if (canvas_height < view_height) then
```

```
    posy = tonumber(ih.posy)
    posy = posy - (move_y/view_height)
  end

  if (canvas_width < view_width) then
    posx = tonumber(ih.posx)
    posx = posx - (move_x/view_width)
  end

  if (posx ~= 0 or posy ~= 0) then
    ih.posx = posx
    ih.posy = posy
    iup.Update(ih)
  end
end

function zoom_update(ih, zoom_index)
  local zoom_lbl = iup.GetDialogChild(ih, "ZOOMLABEL")
  local dlg = iup.GetDialog(ih)
  local canvas = dlg.canvas
  local image = canvas.image
  local zoom_factor = 2^zoom_index

  zoom_lbl.title = string.format("%.0f%%", math.floor(zoom_factor * 100))

  if (image) then
    local view_width = math.floor(zoom_factor * image:Width())
    local view_height = math.floor(zoom_factor * image:Height())

    local old_center_x, old_center_y = scroll_calc_center(canvas)

    scroll_update(canvas, view_width, view_height)

    scroll_center(canvas, old_center_x, old_center_y)
  end
  iup.Update(canvas)
end

function xy_stack_push(q, x, y)
 local new_q = {}
 new_q.x = x
 new_q.y = y
```

```
 new_q.next = q
 return new_q
end

function xy_stack_pop(q)
  local next_q = q.next
  return next_q
end

function color_is_similar(color1, color2, tol)
  local diff_r = cd.Red(color1) - cd.Red(color2)
  local diff_g = cd.Green(color1) - cd.Green(color2)
  local diff_b = cd.Blue(color1) - cd.Blue(color2)
  local sqr_dist = diff_r*diff_r + diff_g*diff_g + diff_b*diff_b
  -- max value = 255*255*3 = 195075
  -- sqrt(195075)=441
  if (sqr_dist < tol) then
    return true
  else
    return false
  end
end

function image_flood_fill(image, start_x, start_y, replace_color, tol_percent)
  local r = image[0]
  local g = image[1]
  local b = image[2]
  local color

  local target_color = cd.EncodeColor(r[start_y][start_x], g[start_y][start_x], b[s

  if (target_color == replace_color) then
    return
  end
  local tol = math.floor((441 * tol_percent) / 100)
  tol = tol*tol  -- this is too high
  tol = tol / 50 -- empirical reduce. TODO: What is the best formula?

  -- very simple 4 neighbors stack based flood fill

  -- a color in the xy_stack is always similar to the target color,
  -- and it was already replaced
```

```
local q = xy_stack_push(nil, start_x, start_y)
r[start_y][start_x], g[start_y][start_x], b[start_y][start_x] = cd.DecodeColor(re

while (q) do
  local cur_x = q.x
  local cur_y = q.y
  q = xy_stack_pop(q)

  -- right
  if (cur_x < image:Width() - 1) then
    color = cd.EncodeColor(r[cur_y][cur_x+1], g[cur_y][cur_x+1], b[cur_y][cur_x+1
    if (color ~= replace_color and color_is_similar(color, target_color, tol)) th
      q = xy_stack_push(q, cur_x+1, cur_y)
      r[cur_y][cur_x+1], g[cur_y][cur_x+1], b[cur_y][cur_x+1] = cd.DecodeColor(re
    end
  end

  -- left
  if (cur_x > 0) then
    color = cd.EncodeColor(r[cur_y][cur_x-1], g[cur_y][cur_x-1], b[cur_y][cur_x-1
    if (color ~= replace_color and color_is_similar(color, target_color, tol)) th
      q = xy_stack_push(q, cur_x-1, cur_y)
      r[cur_y][cur_x-1], g[cur_y][cur_x-1], b[cur_y][cur_x-1] = cd.DecodeColor(re
    end
  end

  -- top
  if (cur_y < image:Height() - 1) then
    color = cd.EncodeColor(r[cur_y+1][cur_x], g[cur_y+1][cur_x], b[cur_y+1][cur_x
    if (color ~= replace_color and color_is_similar(color, target_color, tol)) th
      q = xy_stack_push(q, cur_x, cur_y+1)
      r[cur_y+1][cur_x], g[cur_y+1][cur_x], b[cur_y+1][cur_x] = cd.DecodeColor(re
    end
  end

  -- bottom
  if (cur_y > 0) then
    color = cd.EncodeColor(r[cur_y-1][cur_x], g[cur_y-1][cur_x], b[cur_y-1][cur_x
    if (color ~= replace_color and color_is_similar(color, target_color, tol)) th
      q = xy_stack_push(q, cur_x, cur_y-1)
      r[cur_y-1][cur_x], g[cur_y-1][cur_x], b[cur_y-1][cur_x] = cd.DecodeColor(re
    end
```

```
    end
  end
end

function image_fill_white(image)
  local x, y
  local r = image[0]
  local g = image[1]
  local b = image[2]
  for y = 0, image:Height()-1 do
    local r_line = r[y]
    local g_line = g[y]
    local b_line = b[y]
    for x = 0, image:Width()-1 do
      r_line[x] = 255
      g_line[x] = 255
      b_line[x] = 255
    end
  end
end

function update_image(canvas, image, update_size)
  local old_image = canvas.image

  canvas.dirty = "Yes"
  canvas.image = image

  if (old_image) then
    old_image:Destroy()
  end

  if (update_size) then
    local size_lbl = iup.GetDialogChild(canvas, "SIZELABEL")
    local zoom_val = iup.GetDialogChild(canvas, "ZOOMVAL")
    local zoom_index = zoom_val.value
    size_lbl.title = image:Width().." x ".. image:Height().." px"
    zoom_update(canvas, zoom_index)
  else
    iup.Update(canvas)
  end
end
```

```
function set_new_image(canvas, image, filename, dirty)
  local dlg = iup.GetDialog(canvas)
  local old_image = canvas.image
  local size_lbl = iup.GetDialogChild(canvas, "SIZELABEL")
  local zoom_val = iup.GetDialogChild(canvas, "ZOOMVAL")

  if (filename) then
    canvas.filename = filename
    dlg.title = str_filetitle(filename).." - Simple Paint"
  else
    dlg.title = "Untitled - Simple Paint"
    canvas.filename = nil
  end

  -- we are going to support only RGB images with no alpha
  image:RemoveAlpha()
  if (image:ColorSpace() ~= im.RGB) then
    local new_image = im.ImageCreateBased(image, nil, nil, im.RGB, nil)
    im.ConvertColorSpace(image, new_image)
    image:Destroy()

    image = new_image
  end

  -- default file format
  local format = image:GetAttribString("FileFormat")
  if (not format) then
    image:SetAttribString("FileFormat", "JPEG")
  end

  canvas.dirty = dirty
  canvas.image = image

  size_lbl.title = image:Width().." x "..image:Height().." px"

  if (old_image) then
    old_image:Destroy()
  end

  zoom_val.value = 0
  zoom_update(canvas, 0)
end
```

```
function check_new_file(dlg)
  local canvas = dlg.canvas
  local image = canvas.image
  if (not image) then
    local config = canvas.config
    local width = config:GetVariableDef("NewImage", "Width", 640)
    local height = config:GetVariableDef("NewImage", "Height", 480)

    local image = im.ImageCreate(width, height, im.RGB, im.BYTE)
    if (not image) then
      show_file_error(im.ERR_MEM)
      return
    end

    image_fill_white(image)

    set_new_image(canvas, image, nil, nil)
  end
end

function open_file(ih, filename)
  local image = read_file(filename)
  if (image) then
    local dlg = iup.GetDialog(ih)
    local canvas = dlg.canvas
    local config = canvas.config

    set_new_image(canvas, image, filename, nil)

    config:RecentUpdate(filename)
  end
end

function save_file(canvas)
  if (write_file(canvas.filename, canvas.image)) then
    canvas.dirty = nil
  end
end

function set_file_format(image, filename)
  local ext = str_fileext(filename)
```

```
  ext:lower()
  local format = "JPEG"
  if (ext == "jpg" or ext == "jpeg") then
    format = "JPEG"
  elseif (ext == "bmp") then
    format = "BMP"
  elseif (ext == "png") then
    format = "PNG"
  elseif (ext == "tga") then
    format = "TGA"
  elseif (ext == "tif" or ext == "tiff") then
    format = "TIFF"
  end
  image:SetAttribString("FileFormat", format)
end

function saveas_file(canvas, filename)
  local image = canvas.image

  set_file_format(image, filename)

  if (write_file(filename, image)) then
    local dlg = iup.GetDialog(canvas)
    local config = canvas.config

    dlg.title = str_filetitle(filename).." - Simple Paint"
    canvas.filename = filename
    canvas.dirty = nil

    config:RecentUpdate(filename)
  end
end

function save_check(ih)
  local dlg = iup.GetDialog(ih)
  local canvas = dlg.canvas

  if (canvas.dirty) then
    local resp = iup.Alarm("Warning", "File not saved! Save it now?", "Yes", "No",
    if resp == 1 then -- save the changes and continue
      save_file(canvas)
    elseif resp == 3 then  -- cancel
```

```
      return false
    else  -- ignore the changes and continue
    end
  end
  return true
end

function toggle_bar_visibility(item, bar)
  if (item.value == "ON") then
    bar.floating = "YES"
    bar.visible = "NO"
    item.value = "OFF"
  else
    bar.floating = "NO"
    bar.visible = "YES"
    item.value = "ON"
  end
  iup.Refresh(bar)  -- refresh the dialog layout
end

function select_file(parent_dlg, is_open)
  local filedlg = iup.filedlg{
    extfilter="Image Files|*.bmp;*.jpg;*.png;*.tif;*.tga|All Files|*.*|",
    parentdialog = parent_dlg,
    directory = config:GetVariable("MainWindow", "LastDirectory"),
    }

  if (is_open) then
    filedlg.dialogtype = "OPEN"
  else
    filedlg.dialogtype = "SAVE"
    filedlg.file = canvas.filename
  end

  filedlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(filedlg.status) ~= -1) then
    local filename = filedlg.value
    if (is_open) then
      open_file(parent_dlg, filename)
    else
      saveas_file(canvas, filename)
```

```
    end

    config:SetVariable("MainWindow", "LastDirectory", filedlg.directory)
  end

  filedlg:destroy()
end

function view_fit_rect(canvas_width, canvas_height, image_width, image_height)
  local view_width = canvas_width
  local view_height = (canvas_width * image_height) / image_width

  if (view_height > canvas_height) then
    view_height = canvas_height
    view_width = (canvas_height * image_width) / image_height
  end

  return view_width, view_height
end

function view_zoom_rect(ih, image_width, image_height)
  local zoom_val = iup.GetDialogChild(ih, "ZOOMVAL")
  local zoom_index = tonumber(zoom_val.value)
  local zoom_factor = 2^zoom_index
  local x, y

  local posy = tonumber(ih.posy)
  local posx = tonumber(ih.posx)

  local canvas_width, canvas_height = string.match(ih.drawsize,"(%d*)x(%d*)")
  canvas_width = tonumber(canvas_width)
  canvas_height = tonumber(canvas_height)

  local view_width = math.floor(zoom_factor * image_width)
  local view_height = math.floor(zoom_factor * image_height)

  if (canvas_width < view_width) then
    x = math.floor(-posx * view_width)
  else
    x = math.floor((canvas_width - view_width) / 2)
  end
```

```
  if (canvas_height < view_height) then
    -- posy is top-bottom, CD is bottom-top.
    -- invert posy reference (YMAX-DY - POSY)
    dy = tonumber(canvas.dy)
    posy = 1 - dy - posy
    y = math.floor(-posy * view_height)
  else
    y = math.floor((canvas_height - view_height) / 2)
  end

  return zoom_factor, x, y, view_width, view_height
end

function tool_get_text_enter_cb()
  return iup.CLOSE
end

function tool_get_text()
  local text, dlg

  text = iup.text{expand = "YES", value = toolbox.tooltext, font = toolbox.toolfont

  dlg = iup.dialog{text, title = "Enter Text:", minbox = "NO", maxbox = "NO"}
  dlg.parentdialog = toolbox

  iup.Popup(dlg, iup.MOUSEPOS, iup.MOUSEPOS)

  toolbox.tooltext = text.value

  iup.Destroy(dlg)
end

function tool_draw_pencil(toolbox, image, start_x, start_y, x, y)
  local res = tonumber(iup.GetGlobal("SCREENDPI")) / 25.4

  local line_width = tonumber(toolbox.toolwidth)
  r, g, b = string.match(toolbox.toolcolor, "(%d*) (%d*) (%d*)")

  -- do not use line style here
  local rgb_canvas = cd.CreateCanvas(cd.IMIMAGE, image)
  rgb_canvas:SetAttribute("RESOLUTION", res)
  rgb_canvas:Foreground(cd.EncodeColor(r, g, b))
```

```
  rgb_canvas:LineWidth(line_width)
  rgb_canvas:Line(start_x, start_y, x, y)
  rgb_canvas:Kill()
end

function tool_draw_overlay(toolbox, cnv, start_x, start_y, end_x, end_y)
  local r, g, b = string.match(toolbox.toolcolor, "(%d*) (%d*) (%d*)")
  local line_width = toolbox.toolwidth
  local line_style = toolbox.toolstyle - 1

  cnv:Foreground(cd.EncodeColor(r, g, b))
  cnv:LineWidth(line_width)
  if (line_width == 1) then
    cnv:LineStyle(line_style)
  end

  if (canvas.overlay == "LINE") then
    cnv:Line(start_x, start_y, end_x, end_y)
  elseif (canvas.overlay == "RECT") then
    cnv:Rect(start_x, end_x, start_y, end_y)
  elseif (canvas.overlay == "BOX") then
    cnv:Box(start_x, end_x, start_y, end_y)
  elseif (canvas.overlay == "ELLIPSE") then
    cnv:Arc(math.floor((end_x + start_x) / 2), math.floor((end_y + start_y) / 2), m
  elseif (canvas.overlay == "OVAL") then
    cnv:Sector(math.floor((end_x + start_x) / 2), math.floor((end_y + start_y) / 2)
  elseif (canvas.overlay == "TEXT") then
    cnv:TextAlignment(cd.SOUTH_WEST)
    cnv:NativeFont(toolbox.toolfont)
    cnv:Text(end_x, end_y, toolbox.tooltext)
  end
end


--******************************** Main (Part 1/2) ****************************

-- create all the elements that will have callbacks in Lua prior to callbacks defin

config = iup.config{}
config.app_name = "simple_paint"
config:Load()
```

```
canvas = iup.canvas{
  scrollbar = "Yes",
  dirty = nil, -- custom attribute
  dx = 0,
  dy = 0,
  config = config,
}

item_new = iup.item{title = "&New...\tCtrl+N", image = "IUP_FileNew"}
item_open = iup.item{title = "&Open...\tCtrl+O", image = "IUP_FileOpen"}
item_save = iup.item{title="&Save\tCtrl+S"}
item_saveas = iup.item{title="Save &As...", image = "IUP_FileSave"}
item_revert = iup.item{title="&Revert"}
item_pagesetup = iup.item{title="Page Set&up..."}
item_print = iup.item{title="&Print\tCtrl+P"}
item_exit = iup.item{title="E&xit"}
item_copy = iup.item{title="&Copy\tCtrl+C", image = "IUP_EditCopy"}
item_paste = iup.item{title="&Paste\tCtrl+V", image = "IUP_EditPaste"}
item_background = iup.item{title="&Background..."}
item_zoomin = iup.item{title="Zoom &In\tCtrl++", image = "IUP_ZoomIn"}
item_zoomout = iup.item{title="Zoom &Out\tCtrl+-", image = "IUP_ZoomOut"}
item_actualsize = iup.item{title="&Actual Size\tCtrl+0", image = "IUP_ZoomActualSiz
item_zoomgrid = iup.item{title = "&Zoom Grid", value = "ON"}
item_toolbar = iup.item{title="&Toobar", value="ON"}
item_toolbox = iup.item{title = "&Toobox", value = "ON"}
item_statusbar = iup.item{title="&Statusbar", value="ON"}
item_help = iup.item{title="&Help..."}
item_about = iup.item{title="&About..."}

recent_menu = iup.menu{}

file_menu = iup.menu
{
  item_new,
  item_open,
  item_save,
  item_saveas,
  item_revert,
  iup.separator{},
  item_pagesetup,
  item_print,
  iup.separator{},
```

```
  iup.submenu{title="Recent &Files", recent_menu},
  iup.separator{},
  item_exit,
}

edit_menu = iup.menu
{
  item_copy,
  item_paste,
}

view_menu = iup.menu
{
  item_zoomin,
  item_zoomout,
  item_actualsize,
  item_zoomgrid,
  iup.separator{},
  item_background,
  iup.separator{},
  item_toolbar,
  item_toolbox,
  item_statusbar,
}

image_menu = iup.menu
{
  iup.item{title = "&Resize..."},
  iup.item{title = "&Mirror"},
  iup.item{title = "&Flip"},
  iup.item{title = "&Rotate 180º"},
  iup.item{title = "&Rotate +90º (clock-wise)"},
  iup.item{title = "&Rotate -90º (counter-clock)"},
  iup.separator{},
  iup.item{title = "&Negative"},
  iup.item{title = "&Brightness and Contrast..."},
}

help_menu = iup.menu{item_help, item_about}

sub_menu_file  = iup.submenu{file_menu,  title = "&File"}
sub_menu_edit  = iup.submenu{edit_menu,  title = "&Edit"}
```

```
sub_menu_view  = iup.submenu{view_menu,  title = "&View"}
sub_menu_image = iup.submenu{image_menu, title = "&Image"}
sub_menu_help  = iup.submenu{help_menu,  title = "&Help"}

menu = iup.menu{
  sub_menu_file,
  sub_menu_edit,
  sub_menu_view,
  sub_menu_image,
  sub_menu_help,
  }


--********************************** Callbacks **********************************


function canvas:action()
  local image = canvas.image
  local canvas_width, canvas_height = string.match(canvas.drawsize,"(%d*)x(%d*)")
  local cd_canvas = canvas.cdCanvas

  canvas_width = tonumber(canvas_width)
  canvas_height = tonumber(canvas_height)

  cd_canvas:Activate()

  -- draw the background
  local background = config:GetVariableDef("MainWindow", "Background", "208 208 208
  local r, g, b = string.match(background, "(%d*) (%d*) (%d*)")
  cd_canvas:Background(cd.EncodeColor(r, g, b))
  cd_canvas:Clear()

  -- draw the image at the center of the canvas
  if (image) then
    local zoom_factor, x, y, view_width, view_height = view_zoom_rect(canvas, image

    -- black line around the image
    cd_canvas:Foreground(cd.BLACK)
    cd_canvas:Rect(x - 1, x + view_width, y - 1, y + view_height)

    cd_canvas:PutImImage(image, x, y, view_width, view_height)
```

```
  if (config:GetVariable("MainWindow", "ZoomGrid") == "ON") then
    local zoom_val = iup.GetDialogChild(canvas, "ZOOMVAL")
    local zoom_index = tonumber(zoom_val.value)
    if (zoom_index > 1) then
      local ix, iy
      local zoom_factor = math.pow(2, zoom_index)
      cd_canvas:Foreground(cd.GRAY)
      for ix = 0, image:Width()-1 do
        local gx = math.floor(ix * zoom_factor)
        cd_canvas:Line(gx + x, y, gx + x, y + view_height)
      end
      for iy = 0, image:Height()-1 do
        local gy = math.floor(iy * zoom_factor)
        cd_canvas:Line(x, gy + y, x + view_width, gy + y)
      end
    end
  end

  if (canvas.overlay) then
    local start_x = tonumber(canvas.start_x)
    local start_y = tonumber(canvas.start_y)
    local end_x = tonumber(canvas.end_x)
    local end_y = tonumber(canvas.end_y)

    local scale_x = view_width / image:Width()
    local scale_y = view_height / image:Height()

    -- offset and scale drawing in screen to macth the image
    if (scale_x > 1 or scale_y > 1) then
      -- also draw at the center of the pixel when zoom in
      cd_canvas:TransformTranslate(x + scale_x / 2, y + scale_y / 2)
    else
      cd_canvas:TransformTranslate(x, y)
    end
    cd_canvas:TransformScale(scale_x, scale_y)

    tool_draw_overlay(toolbox, cd_canvas, start_x, start_y, end_x, end_y)

    cd_canvas:Transform(nil)
  end
end
cd_canvas:Flush()
```

```
end

function canvas:map_cb()
  cd_canvas = cd.CreateCanvas(cd.IUPDBUFFER, canvas)
  canvas.cdCanvas = cd_canvas
end

function canvas:unmap_cb()
  local cd_canvas = canvas.cdCanvas
  cd_canvas:Kill()
end

function round(x)
  if (x < 0) then
    return math.ceil(x - 0.5)
  else
    return math.floor(x + 0.5)
  end
end

function item_zoomout:action()
  local zoom_val = iup.GetDialogChild(self, "ZOOMVAL")
  local zoom_index = tonumber(zoom_val.value)
  zoom_index = zoom_index - 1
  if (zoom_index < -6) then
    zoom_index = -6
  end
  zoom_val.value = round(zoom_index)  -- fixed increments when using buttons

  zoom_update(self, zoom_index)
end

function item_zoomin:action()
  local zoom_val = iup.GetDialogChild(self, "ZOOMVAL")
  local zoom_index = tonumber(zoom_val.value)
  zoom_index = zoom_index + 1
  if (zoom_index > 6) then
    zoom_index = 6
  end
  zoom_val.value = round(zoom_index)  -- fixed increments when using buttons

  zoom_update(self, zoom_index)
```

```
end

function item_actualsize:action()
  local zoom_val = iup.GetDialogChild(self, "ZOOMVAL")
  zoom_val.value = 0
  zoom_update(self, 0)
end

function canvas:resize_cb()
  local image = canvas.image
  if (image) then
    local zoom_val = iup.GetDialogChild(self, "ZOOMVAL")
    local zoom_index = tonumber(zoom_val.value)
    local zoom_factor = 2^zoom_index

    local view_width = math.floor(zoom_factor * image:Width())
    local view_height = math.floor(zoom_factor * image:Height())

    local old_center_x, old_center_y = scroll_calc_center(canvas)

    scroll_update(canvas, view_width, view_height)

    scroll_center(canvas, old_center_x, old_center_y)
  end
end

function canvas:wheel_cb(delta)
  if (iup.GetGlobal("CONTROLKEY") == "ON") then
    if (delta < 0) then
      item_zoomout:action()
    else
      item_zoomin:action()
    end
  else
    local posy = tonumber(canvas.posy)
    posy = posy - delta * tonumber(canvas.dy) / 10
    canvas.posy = posy
    iup.Update(canvas)
  end
end

function canvas:button_cb(button, pressed, x, y)
```

```
local image = self.image
if (image) then
  local cursor_x, cursor_y = x, y
  local zoom_factor, view_x, view_y, view_width, view_height = view_zoom_rect(can

  -- y is top-down in IUP
  local canvas_width, canvas_height = string.match(canvas.drawsize,"(%d*)x(%d*)")
  canvas_height = tonumber(canvas_height)
  y = canvas_height - y - 1

  -- inside image area
  if (x > view_x and y > view_y and x < view_x + view_width and y < view_y + view
    x = x - view_x
    y = y - view_y

    x = math.floor(x / zoom_factor)
    y = math.floor(y / zoom_factor)

    if (x < 0) then x = 0 end
    if (y < 0) then y = 0 end
    if (x > image:Width() - 1) then x = image:Width() - 1 end
    if (y > image:Height() - 1) then y = image:Height() - 1 end

    if (button == iup.BUTTON1) then
      if (pressed == 1) then
        canvas.start_x = x
        canvas.start_y = y
        canvas.start_cursor_x = cursor_x
        canvas.start_cursor_y = cursor_y
      else
        local tool_index = tonumber(toolbox.toolindex)
        if (tool_index == 1) then -- Color Picker
          local color = toolbox.color
          r = image[0][y][x]
          g = image[1][y][x]
          b = image[2][y][x]
          color.bgcolor = r.."␣"..g.."␣"..b
          toolbox.toolcolor = r.."␣"..g.."␣"..b
        elseif (tool_index == 2) then -- Pencil
          local start_x = canvas.start_x
          local start_y = canvas.start_y
```

```
        tool_draw_pencil(toolbox, image, start_x, start_y, x, y)

        canvas.dirty = "Yes"
        iup.Update(canvas)

        canvas.start_x = x
        canvas.start_y = y
      elseif (tool_index >= 3 and tool_index <= 8) then -- Shapes
        if (canvas.overlay) then
          local start_x = canvas.start_x
          local start_y = canvas.start_y
          local res = tonumber(iup.GetGlobal("SCREENDPI")) / 25.4

          local rgb_canvas = cd.CreateCanvas(cd.IMIMAGE, image)
          rgb_canvas:SetAttribute("RESOLUTION", res)

          tool_draw_overlay(toolbox, rgb_canvas, start_x, start_y, x, y)

          rgb_canvas:Kill()

          canvas.overlay = nil
          canvas.dirty = "Yes"

          iup.Update(canvas)
        end
      elseif (tool_index == 9) then -- Fill Color
        local tol_percent = toolbox.toolfilltol
        local r, g, b = string.match(toolbox.toolcolor, "(%d*) (%d*) (%d*)")
        image_flood_fill(image, x, y, cd.EncodeColor(r, g, b), tol_percent)
        canvas.dirty = "Yes"

        iup.Update(canvas)
      end
    end
  elseif (button == iup.BUTTON3) then
    if (pressed == 0) then
      local tool_index = tonumber(toolbox.toolindex)
      if (tool_index == 8) then -- Text
        tool_get_text()
      end
    end
  end
```

```
    end
  end
  return iup.DEFAULT
end

function canvas:motion_cb(x, y, status)
  local image = self.image
  if (image) then
    local cursor_x, cursor_y = x, y
    local zoom_factor, view_x, view_y, view_width, view_height = view_zoom_rect(can

    -- y is top-down in IUP
    local canvas_width, canvas_height = string.match(canvas.drawsize,"(%d*)x(%d*)")
    canvas_height = tonumber(canvas_height)
    y = canvas_height - y - 1

    -- inside image area
    if (x > view_x and y > view_y and x < view_x + view_width and y < view_y + view
      local status_lbl = iup.GetDialogChild(self, "STATUSLABEL")

      x = x - view_x
      y = y - view_y

      x = math.floor(x / zoom_factor)
      y = math.floor(y / zoom_factor)

      if (x < 0) then x = 0 end
      if (y < 0) then y = 0 end
      if (x > image:Width() - 1) then x = image:Width() - 1 end
      if (y > image:Height() - 1) then y = image:Height() - 1 end

      local r = image[0][y][x]
      local g = image[1][y][x]
      local b = image[2][y][x]

      status_lbl.title = "("..x..", "..y..") = "..r.." "..g.." "..b

      if (iup.isbutton1(status)) then -- button1 is pressed
        local tool_index = tonumber(toolbox.toolindex)

        if (tool_index == 0) then -- Pointer
          if not canvas.start_cursor_x then canvas.start_cursor_x = cursor_x end
```

```
            if not canvas.start_cursor_y then canvas.start_cursor_y = cursor_y end
            local start_cursor_x = tonumber(canvas.start_cursor_x)
            local start_cursor_y = tonumber(canvas.start_cursor_y)

            local canvas_width, canvas_height = string.match(self.drawsize,"(%d*)x(%d
            canvas_width = tonumber(canvas_width)
            canvas_height = tonumber(canvas_height)

            scroll_move(canvas, canvas_width, canvas_height, cursor_x - start_cursor_x

            self.start_cursor_x = cursor_x
            self.start_cursor_y = cursor_y
          elseif (tool_index == 2) then -- Pencil
            local start_x = self.start_x
            local start_y = self.start_y

            tool_draw_pencil(toolbox, image, start_x, start_y, x, y)

            self.dirty = "Yes"
            iup.Update(canvas)

            self.start_x = tonumber(x)
            self.start_y = tonumber(y)
          elseif (tool_index >= 3 and tool_index <= 8) then -- Shapes
            local shapes = {"LINE", "RECT", "BOX", "ELLIPSE", "OVAL", "TEXT"}
            self.end_x = tonumber(x)
            self.end_y = tonumber(y)
            self.overlay = shapes[tool_index - 2]
            iup.Update(self)
          end
        end
      end
    end
  end
end

function zoom_valuechanged_cb(val)
  local zoom_index = tonumber(val.value)
  zoom_update(val, zoom_index)
end

function canvas:dropfiles_cb(filename)
  if (save_check(self)) then
```

```
    open_file(self, filename)
  end
end

function file_menu:open_cb()
  if (canvas.dirty) then
    item_save.active = "YES"
  else
    item_save.active = "NO"
  end
  if (canvas.dirty and canvas.filename) then
    item_revert.active = "YES"
  else
    item_revert.active = "NO"
  end
end

function edit_menu:open_cb()
  local clipboard = iup.clipboard{}
  if (clipboard.imageavailable == "NO") then
    item_paste.active = "NO"
  else
    item_paste.active = "YES"
  end
  clipboard:destroy()
end

function config:recent_cb()
  if (save_check(self)) then
    local filename = self.title
    open_file(self, filename)
  end
end

function item_new:action()
  if save_check(self) then
    local width = config:GetVariableDef("NewImage", "Width", 640)
    local height = config:GetVariableDef("NewImage", "Height", 480)

    local ret, new_width, new_height = iup.GetParam("New␣Image", nil, "Width:␣%i[1,
    if (ret) then
      local new_image = im.ImageCreate(new_width, new_height, im.RGB, im.BYTE)
```

```
      if (not new_image) then
        show_file_error(im.ERR_MEM)
        return
      end

      config:SetVariable("NewImage", "Width", new_width)
      config:SetVariable("NewImage", "Height", new_height)

      set_new_image(canvas, new_image, nil, nil)
    end
  end
end

function item_open:action()
  if not save_check(self) then
    return
  end

  select_file(dlg, true)
end

function item_saveas:action()
  select_file(dlg, false)
end

function item_save:action()
  if (not canvas.filename) then
    item_saveas:action()
  else
    -- test again because in can be called using the hot key
    if (canvas.dirty) then
      save_file(canvas)
    end
  end
end

function item_revert:action()
  open_file(self, canvas.filename)
end

function item_pagesetup:action()
  local width = config:GetVariableDef("Print", "MarginWidth", 20)
```

```
  local height = config:GetVariableDef("Print", "MarginHeight", 20)

  local ret, new_width, new_height = iup.GetParam("Page Setup", nil, "nMargin Width
  if (ret) then
    config:SetVariable("Print", "MarginWidth", new_width)
    config:SetVariable("Print", "MarginHeight", new_height)
  end
end

function item_print:action()
  local title = dlg.title
  local print_canvas = cd.CreateCanvas(cd.PRINTER, title.." -d")
  if (not print_canvas) then
    return
  end

  -- do NOT draw the background, use the paper color

  -- draw the image at the center of the canvas
  local image = canvas.image
  if (image) then
    local margin_width = config:GetVariableDef("Print", "MarginWidth", 20)
    local margin_height = config:GetVariableDef("Print", "MarginHeight", 20)

    local canvas_width, canvas_height, canvas_width_mm, canvas_height_mm = print_ca

    -- convert to pixels
    margin_width = math.floor((margin_width * canvas_width) / canvas_width_mm)
    margin_height = math.floor((margin_height * canvas_height) / canvas_height_mm)

    local view_width, view_height = view_fit_rect(
       canvas_width - 2 * margin_width, canvas_height - 2 * margin_height,
       image:Width(), image:Height())

    local x = math.floor((canvas_width - view_width) / 2)
    local y = math.floor((canvas_height - view_height) / 2)

    print_canvas:PutImImage(image, x, y, view_width, view_height)
  end

  print_canvas:Kill()
end
```

```
function item_exit:action()
  local image = canvas.image

  if not save_check(self) then
    return iup.IGNORE  -- to abort the CLOSE_CB callback
  end

  if (toolbox.visible == "YES") then
    config:DialogClosed(toolbox, "Toolbox")
    toolbox:hide()
  end

  if (image) then
    image:Destroy()
    canvas.image = nil
  end

  config:DialogClosed(iup.GetDialog(self), "MainWindow")
  config:Save()
  config:destroy()
  return iup.CLOSE
end

function item_copy:action()
  local clipboard = iup.clipboard{}
  -- must use iup.SetAttribute because it is an userdata
  iup.SetAttribute(clipboard, "NATIVEIMAGE", iup.GetImageNativeHandle(canvas.image))
  clipboard:destroy()
end

function item_paste:action()
  if save_check(self) then
    local clipboard = iup.clipboard{}
    local image = iup.GetNativeHandleImage(clipboard.nativeimage)
    clipboard:destroy()

    if (not image) then
      show_error("Invalid Clipboard Data", true)
      return
    end
```

```
    set_new_image(canvas, image, nil, "Yes")
  end
end

function item_background:action()
  local colordlg = iup.colordlg{}
  local background = config:GetVariableDef("MainWindow", "Background", "255 255 255
  colordlg.value = background
  colordlg.parentdialog = iup.GetDialog(self)

  colordlg:popup(iup.CENTERPARENT, iup.CENTERPARENT)

  if (tonumber(colordlg.status) == 1) then
    background = colordlg.value
    config:SetVariable("MainWindow", "Background", background)

    iup.Update(canvas)
  end

  colordlg:destroy()
end

function item_zoomgrid:action()
  if (self.value == "ON") then
    self.value = "OFF"
  else
    self.value = "ON"
  end
  config:SetVariable("MainWindow", "ZoomGrid", self.value)

  iup.Update(canvas)
end

function item_toolbar:action()
  toggle_bar_visibility(self, toolbar)
  config:SetVariable("MainWindow", "Toolbar", item_toolbar.value)
end

function item_toolbox:action()
  if (toolbox.visible == "YES") then
    self.value = "OFF"
    config:DialogClosed(toolbox, "Toolbox")
```

```
    iup.Hide(toolbox)
  else
    self.value = "ON"
    config:DialogShow(toolbox, "Toolbox")
  end
  config:SetVariable("MainWindow", "Toolbox", self.value)
end

function item_statusbar:action()
  toggle_bar_visibility(self, statusbar)
  config:SetVariable("MainWindow", "Statusbar", item_statusbar.value)
end

item_resize = image_menu[1]
function item_resize:action()
  local image = canvas.image
  local quality = config:GetVariableDef("Image", "ResizeQuality", 1)  -- medium def
  local width = image:Width()
  local height = image:Height()

  local ret, new_width, new_height = iup.GetParam("Resize", nil, "Width: %i[1,]\nHe
  if (ret) then
    local new_image = im.ImageCreateBased(image, new_width, new_height, nil, nil)
    if (not new_image) then
      show_file_error(im.ERR_MEM)
      return
    end

    config:SetVariable("Image", "ResizeQuality", quality)

    if (quality == 2) then
      quality = 3 -- interpolation order can be 0, 1, and 3
    end

    im.ProcessResize(image, new_image, quality)

    update_image(canvas, new_image, true)  -- update size
  end
end

item_mirror = image_menu[2]
function item_mirror:action()
```

```
  local image = canvas.image
  local new_image = image:Clone()
  if (not new_image) then
    show_file_error(im.ERR_MEM)
    return
  end

  im.ProcessMirror(image, new_image)

  update_image(canvas, new_image, false)
end

item_flip = image_menu[3]
function item_flip:action()
  local image = canvas.image
  local new_image = image:Clone()
  if (not new_image) then
    show_file_error(im.ERR_MEM)
    return
  end

  im.ProcessFlip(image, new_image)

  update_image(canvas, new_image, false)
end

item_rotate180 = image_menu[4]
function item_rotate180:action()
  local image = canvas.image
  local new_image = image:Clone()
  if (not new_image) then
    show_file_error(im.ERR_MEM)
    return
  end

  im.ProcessRotate180(image, new_image)

  update_image(canvas, new_image, false)
end

item_rotate90cw = image_menu[5]
function item_rotate90cw:action()
```

```
  local image = canvas.image
  local new_image = im.ImageCreateBased(image, image:Height(), image:Width(), nil,
  if (not new_image) then
    show_file_error(im.ERR_MEM)
    return
  end

  im.ProcessRotate90(image, new_image, 1)

  update_image(canvas, new_image, true)  -- update size
end

item_rotate90ccw = image_menu[6]
function item_rotate90ccw:action()
  local image = canvas.image
  local new_image = im.ImageCreateBased(image, image:Height(), image:Width(), nil,
  if (not new_image) then
    show_file_error(im.ERR_MEM)
    return
  end

  im.ProcessRotate90(image, new_image, -1)

  update_image(canvas, new_image, true)  -- update size
end

item_negative = image_menu[8] -- skip separator
function item_negative:action()
  local image = canvas.image
  local new_image = image:Clone()
  if (not new_image) then
    show_file_error(im.ERR_MEM)
    return
  end

  im.ProcessNegative(image, new_image)

  update_image(canvas, new_image, false)
end

function brightcont_param_cb(dialog, param_index)
  if (param_index == 0 or param_index == 1) then
```

```
    local image = canvas.original_image
    local new_image = canvas.new_image
    local brightness_shift_param = iup.GetParamParam(dialog, 0)
    local contrast_factor_param = iup.GetParamParam(dialog, 1)

    param = { 0, 0 }
    param[1] = tonumber(brightness_shift_param.value)
    param[2] = tonumber(contrast_factor_param.value)

    im.ProcessToneGamut(image, new_image, im.GAMUT_BRIGHTCONT, param)

    canvas.image = canvas.new_image
    iup.Update(canvas)
  elseif (param_index ~= iup.GETPARAM_INIT) then
    -- restore original configuration
    canvas.image = canvas.original_image
    canvas.original_image = nil
    canvas.new_image = nil

    if (param_index == iup.GETPARAM_BUTTON2) then  -- cancel
      iup.Update(canvas)
    end
  end

  return 1
end

item_brightcont = image_menu[9]
function item_brightcont:action()
  local image = canvas.image
  local new_image = image:Clone()
  if (not new_image) then
    show_file_error(im.ERR_MEM)
    return
  end

  local param = { 0, 0 }

  canvas.original_image = image
  canvas.new_image = new_image

  ret, param[1], param[2] = iup.GetParam("Brightness and Contrast", brightcont_param
```

```
  if (not ret) then
    new_image:Destroy()
          return
  end

  im.ProcessToneGamut(image, new_image, im.GAMUT_BRIGHTCONT, param)

  update_image(canvas, new_image, false)
end

function item_help:action()
  iup.Help("http://www.tecgraf.puc-rio.br/iup")
end

function item_about:action()
  iup.Message("About", "␣␣␣Simple␣Paint\n\nAutors:\n␣␣␣Gustavo␣Lyrio\n␣␣␣Antonio␣Sc
end

function tool_action_cb(ih, state)
  if (state == 1) then
    local tool_index = tonumber(ih.toolindex)
    toolbox.toolindex = tool_index

    if (tool_index == 0) then
      canvas.cursor = "ARROW"
    else
      canvas.cursor = "CROSS"
    end
    if (tool_index == 8) then
      tool_get_text()
    end
  end
end

function toolcolor_action_cb(self)
  local colordlg = iup.colordlg{}
  local color = self.bgcolor
  colordlg.value = color
  colordlg.parentdialog = iup.GetDialog(self)

  colordlg:popup(iup.CENTER, iup.CENTER)
```

```
  if (tonumber(colordlg.status) == 1) then
    color = colordlg.value
    self.bgcolor = color
    toolbox.toolcolor = color

    iup.Update(canvas)
  end

  colordlg:destroy()
end

function toolwidth_valuechanged_cb(self)
  local value = self.value
  iup.SetAttribute(iup.GetDialog(self), "TOOLWIDTH", value)
end

function toolstyle_valuechanged_cb(self)
  local value = self.value
  iup.SetAttribute(iup.GetDialog(self), "TOOLSTYLE", value)
end

function toolfont_action_cb(self)
  local font_dlg = iup.FontDlg()
  font_dlg.parentdialog = iup.GetDialog(self)
  local font = self.toolfont
  font_dlg.value = font

  font_dlg:Popup(iup.CENTER, iup.CENTER)

  if (font_dlg.status == 1) then
    font = font_dlg.value
    iup.SetAttribute(iup.GetDialog(self), "TOOLFONT", font)
  end
  fond_dlg:destroy()
end

function toolfilltol_valuechanged_cb(self)
  local filltol_label = iup.GetDialogChild(self, "FILLTOLLABEL")
  local value = self.value
  filltol_label.title = "Tol.: " .. value .. "%"
  toolbox.toolfilltol = value
end
```

```
--********************************* Main (Part 2/2) ****************************


-- toolbox
gbox = iup.gridbox{
  iup.toggle{toolindex=0, image=load_image_PaintPointer(), value="ON", flat="Yes",
  iup.toggle{toolindex=1, image=load_image_PaintColorPicker(), flat="Yes", tip="Col
  iup.toggle{toolindex=2, image=load_image_PaintPencil(), flat="Yes", tip="Pencil",
  iup.toggle{toolindex=3, image=load_image_PaintLine(), flat="Yes", tip="Line", act
  iup.toggle{toolindex=4, image=load_image_PaintRect(), flat="Yes", tip="Hollow␣Rec
  iup.toggle{toolindex=5, image=load_image_PaintBox(), flat="Yes", tip="Box␣(Filled
  iup.toggle{toolindex=6, image=load_image_PaintEllipse(), flat="Yes", tip="Hollow␣
  iup.toggle{toolindex=7, image=load_image_PaintOval(), flat="Yes", tip="Oval␣(Fille
  iup.toggle{toolindex=8, image=load_image_PaintText(), flat="Yes", tip="Text", act
  iup.toggle{toolindex=9, image=load_image_PaintFill(), flat="Yes", tip="Fill␣Color
  gapcol = 2,
  gaplin = 2,
  margin = "5x10",
  numdiv = 2
}

vbox = iup.vbox{
  iup.radio{gbox},
  iup.frame{
    iup.vbox{
      iup.label{title = "Color:", expand="HORIZONTAL"},
      iup.button{name="COLOR", bgcolor="0␣0␣0", rastersize="28x21", action = toolcol
      iup.label{title="Width:", expand="HORIZONTAL"},
      iup.text{SPIN="Yes", spinmin=1, rastersize="48x", valuechanged_cb = toolwidth_
      iup.label{title="Style:", expand="HORIZONTAL"},
      iup.list{"____", "----", ". . . .", "-.-.", "-..-..", dropdown="Yes", value=1, v
      iup.label{title = "Tol.:␣50%", expand="HORIZONTAL", name="FILLTOLLABEL"},
      iup.val{name="FILLTOL", rastersize="60x30", value=50, max=100, valuechanged_c
      iup.label{title="Font:", expand="HORIZONTAL"},
      iup.button{title = "F", name="FONT", rastersize="21x21", font="Times,␣Bold␣It
      margin="3x2",
      gap=2,
      alignment="ACENTER"
    }
  },
```

```
  nmargin = "2x2",
  aligment = "ACENTER"
}

toolbox = iup.dialog{
  vbox,
  dialogframe = "Yes",
  title = "Tools",
  fontsize = 8,
  toolbox = "Yes",
  -- custom attributes
  toolcolor = "0 0 0",
  toolwidth = 1,
  toolstyle = 1,
  toolfilltol = 50,
  toolindex = 0,
}

--toolbar
btn_new = iup.button{image = "IUP_FileNew", flat = "Yes", action = item_new.action,
btn_open = iup.button{image = "IUP_FileOpen", flat = "Yes", action = item_open.actio
btn_save = iup.button{image = "IUP_FileSave", flat = "Yes", action = item_save.actio
btn_copy = iup.button{image =  "IUP_EditCopy", flat = "Yes", action = item_copy.act
btn_paste = iup.button{image = "IUP_EditPaste", flat = "Yes", action = item_paste.a
btn_zoomgrid = iup.button{image = load_image_PaintZoomGrid(), flat = "Yes", action =

toolbar = iup.hbox{
  btn_new,
  btn_open,
  btn_save,
  iup.label{separator="VERTICAL"},
  btn_copy,
  btn_paste,
  iup.label{separator="VERTICAL"},
  btn_zoomgrid,
  margin = "5x5",
  gap = 2,
}

function toolbox:close_cb()
  config:DialogClosed(self, "Toolbox")
  item_toolbox.value = "OFF"
```

```
  config:SetVariableStr("MainWindow", "Toolbox", "OFF")
end

--statusbar
statusbar = iup.hbox{
  iup.label{title = "(0, 0) = 0 0 0", expand="HORIZONTAL", padding="10x5", name
  iup.label{separator="VERTICAL"},
  iup.label{title = "0 x 0", size="70x", padding="10x5", name="SIZELABEL", alignmen
  iup.label{SEPARATOR="VERTICAL"},
  iup.label{title = "100%", size="30x", padding="10x5", name="ZOOMLABEL", alignment=
  iup.button{IMAGE="IUP_ZoomOut", flat="Yes", tip="Zoom Out (Ctrl+-)", action = iten
  iup.val{value=0, min=-6, max=6, rastersize="150x25", name="ZOOMVAL", valuechanged_
  iup.button{image="IUP_ZoomIn", flat="Yes", tip="Zoom In (Ctrl++)", action = item_
  iup.button{image="IUP_ZoomActualSize", flat="Yes", tip="Actual Size (Ctrl+0)", ac
  alignment = "ACENTER",
}

vbox = iup.vbox{
  toolbar,
  canvas,
  statusbar
}

dlg = iup.dialog{
  vbox,
  title = "Simple Paint",
  size = "HALFxHALF",
  menu = menu,
  close_cb = item_exit.action,
  canvas = canvas,
  dropfiles_cb = canvas.dropfiles_cb,
}

-- must be set after dlg was created
toolbox.parentdialog = dlg
toolbox.toolfont = dlg.font

function dlg:move_cb(x, y)
  local old_x = tonumber(self._old_x)
  local old_y = tonumber(self._old_y)
  if (old_x == x and old_y == y) then
    return
```

```
  end
  if (toolbox.visible == "YES") then
    local tb_x = tonumber(toolbox.x)
    local tb_y = tonumber(toolbox.y)

    tb_x = tb_x + x - old_x
    tb_y = tb_y + y - old_y

    toolbox:showxy(tb_x, tb_y)
  end

  self._old_x = x
  self._old_y = y
end

function dlg:k_any(c)
  if (c == iup.K_cN) then
    item_new:action()
  elseif (c == iup.K_cO) then
    item_open:action()
  elseif (c == iup.K_cS) then
    item_save:action()
  elseif (c == iup.K_cV) then
    item_paste:action()
  elseif (c == iup.K_cC) then
    item_copy:action()
  elseif (c == iup.K_cP) then
    item_print:action()
  elseif (c == iup.K_cMinus) then
    item_zoomout:action()
  elseif (c == iup.K_cPlus or c == iup.K_cEqual) then
    item_zoomin:action()
  elseif (c == iup.K_c0) then
    item_actualsize:action()
  end
end

-- parent for pre-defined dialogs in closed functions (IupMessage and IupAlarm)
iup.SetGlobal("PARENTDIALOG", iup.SetHandleName(dlg))

-- Initialize variables from the configuration file
```

```
config:RecentInit(recent_menu, 10)

local show_zoomgrid = config:GetVariableDef("MainWindow", "ZoomGrid", "ON")
if (show_zoomgrid == "OFF") then
  item_zoomgrid.value = "OFF"
end

local show_statusbar = config:GetVariableDef("MainWindow", "Statusbar", "ON")
if (show_statusbar == "OFF") then
  item_statusbar.value = "OFF"
  statusbar.floating = "YES"
  statusbar.visible = "NO"
end

local show_toolbar = config:GetVariableDef("MainWindow", "Toolbar", "ON")
if (show_toolbar == "OFF") then
  item_toolbar.value = "OFF"
  toolbar.floating = "YES"
  toolbar.visible = "NO"
end

-- show the dialog at the last position, with the last size
config:DialogShow(dlg, "MainWindow")

local show_toolbox = config:GetVariableDef("MainWindow", "Toolbox", "ON")
if (show_toolbox == "ON") then
  -- configure the very first time to be aligned with the main window
  if (not config:GetVariable("Toolbox", "X")) then
    config:SetVariable("Toolbox", "X", canvas.x)
    config:SetVariable("Toolbox", "Y", canvas.y)
  end

  config:DialogShow(toolbox, "Toolbox")
else
  item_toolbox.value = "OFF"
end

-- open a file from the command line (allow file association in Windows)
if (arg and arg[1]) then
  filename = arg[1]
  open_file(dlg, filename)
end
```

```
-- initialize the current file, if not already loaded
check_new_file(dlg)

-- to be able to run this script inside another context
if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
  iup.Close()
end
```

在下一章，我们将 **IUP** 程序设计的一些高级技术。

5 高级主题

5.1 C++ 封装

可以看到，我们第四章的绘图程序已经有 2500 行的代码。如果不是一步步地修改，可能我们就很难理解这些代码。所以，现在是时候使用软件工程的方法来提供我们程序的质量和可维护性。我们将会使用 C++，虽然 C 也可以，但是 C++ 提供了更多的工具，使这项工作变得更加轻松。我们首先需要把 C 代码转换为 C++ 代码。

实际上，简单地重命名源代码文件从.c 到.cpp，就可以使用 C++ 编译器编译通过。但这不是我们认为的使用 C++。我们将会把代码尽量分离，使每部分之间的相互影响尽可能的小。这在软件工程领域也被叫做封装。最简单的进行这项工作的方式就是使用类来组织程序。观察我们的代码，至少有 3 大组功能：主窗口，工具栏窗口，文件管理。

所以在这里，我们是首先创建三个类：*SimplePaint*[10], *SimplePaintToolbox*[11] 和 *SimplePaintFile*[12]。比较 C 和 C++ 代码会发现，它们非常相似，除了原来的函数现在用类方法组织。甚至 **IUP** 回调也是方法，但需要注意类方法的指针不能作为函数指针给 *IupSetCallback* 函数使用。我们必须把它定义成静态形式，然后才能被 *IupSetCallback* 函数使用。在这里我们定义一个宏还简化定义用于回调的类方法。

这些宏被定义在 *iup_class_cbs.hpp* 中。使用这些宏，必须先调用 *IUP_CLASS_INITCALLBACK(ih, class)* 宏，通常我们在类的构造函数中的对话框创建之后调用它。这个宏会注册 **IUP** 元素来使这些元素可以方便地被程序在以后获取。我们使用 *IUP_CLASS_DECLARECALLBACK_*(class,callback)* 来定义回调方法。由于我们有不同类型的回调，它们具有不同的参数，所以我们需要不同的宏来定义。

```
class SampleClass
{
  int sample_count;

public:
  SampleClass()
  {
```

[10] 主窗口。
[11] 工具栏。
[12] 文件管理。

```
    sample_count = 0;

    Ihandle* button = IupButton("Inc", NULL);
    // 2) Associate the callback with the button
    IUP_CLASS_SETCALLBACK(button, "ACTION", ButtonAction);

    Ihandle* dialog = IupDialog(button);
    // 1) Register this object as a callback receiver (only once)
    IUP_CLASS_INITCALLBACK(dialog, SampleClass);

    IupShow(dialog);
  };

protected:
  // 3) Declare the callback as a member function
  IUP_CLASS_DECLARECALLBACK_IFn(SampleClass, ButtonAction);
};

// 4) Define the callback as a member function
int SampleClass::ButtonAction(Ihandle*)
{
  sample_count++;
  return IUP_DEFAULT;
}
```

5.2 C++ 模块

5.3 高清显示

5.4 闪屏，关于和系统信息

5.5 动态链接库

6 7GUIs

7GUIs 是用来评估图形界面工具包的 7 个任务。

这 7 个任务同样也可以作为图形界面程序开发入门的不错选择。

6.1 计数器

挑战: 理解工具包的基本理念。

这个任务要求我们创建一个窗体，窗体上放置一个标签或只读的文本域 T，以及一个按钮 B。刚开始 T 的值为 0，每一次点击按钮 B，T 的值就加 1。

```
#include <stdlib.h>
#include <iup.h>

int btn_count_cb( Ihandle *self )
{
  Ihandle* text = IupGetDialogChild(self, "TEXT");
  int value = IupGetInt(text, "VALUE");
  IupSetInt(text, "VALUE", ++value);

  return IUP_DEFAULT;
}

int main(int argc, char **argv)
{
  Ihandle *dlg, *button, *text, *hbox;

  IupOpen(&argc, &argv);

  button = IupButton("Count", NULL);
  IupSetAttribute(button, "SIZE", "60");
  text = IupText(NULL);
  IupSetAttribute(text, "SIZE", "60");
  IupSetAttribute(text, "NAME", "TEXT");
  IupSetAttribute(text, "READONLY", "YES");
  hbox = IupHbox(
    text,
    button,
    NULL);
  IupSetAttribute(hbox, "MARGIN", "10x10");
  IupSetAttribute(hbox, "GAP", "10");
  dlg = IupDialog(hbox);
  IupSetAttribute(dlg, "TITLE", "Counter");

  IupSetInt(text, "VALUE", 0);

  /* Registers callbacks */
  IupSetCallback(button, "ACTION", (Icallback) btn_count_cb);

  IupShowXY(dlg, IUP_CENTER, IUP_CENTER);

  IupMainLoop();
```

```
  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")

counter = 0

function addCount()
        counter = counter + 1
end

function getCount()
        return counter
end

--********************************** Main *****************************************

txt_count = iup.text{value = getCount(), readonly = "YES",  size = "60"}
btn_count = iup.button{title = "Count", size = "60"}

dlg = iup.dialog{iup.hbox{txt_count, btn_count; ngap = "10"}, title = "Counter", ma

function btn_count:action()
  addCount()
  txt_count.value = getCount()
end

dlg:showxy( iup.CENTER, iup.CENTER )

if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
end
```

还可以使用 *IupText* 元素的 *SPIN* 属性来允许用户增加或减少数值。

6.2 温度转换

挑战: *双向数据流，用户文本输入。*

这个任务是创建一个窗口，包含文本域 *TC* 和文本域 *TF* 分别表示摄氏温度和华氏温度。开始时，*TC* 和 *TF* 都是空的。当用户向 *TC* 中输入数字，*TF* 中的数字会自动更新，反之亦然。当用户向文本域输入非数字值时，不会更新另一个文本域。这两种温度单位的转换公式是：$C = (F - 32) * (5/9)$ 和 $F = C * (9/5) + 32$。

C语言

```
#include <stdlib.h>
#include <iup.h>

int txt_celcius_cb(Ihandle *self)
{
  Ihandle* fahrenheit = IupGetDialogChild(self, "FAHRENHEIT");
  double value = IupGetDouble(self, "VALUE");;

  value = value * (9./5.) + 32;

  IupSetStrf(fahrenheit, "VALUE", "%.2lf", value);

  return IUP_DEFAULT;
}

int txt_fahrenheit_cb(Ihandle *self)
{
  Ihandle* celcius = IupGetDialogChild(self, "CELCIUS");
  double value = IupGetDouble(self, "VALUE");;

  value = (value - 32) * (5./9.);

  IupSetStrf(celcius, "VALUE", "%.2lf", value);

  return IUP_DEFAULT;
}

int main(int argc, char **argv)
{
  Ihandle *dlg, *labelC, *labelF, *tempC, *tempF, *hbox;

  IupOpen(&argc, &argv);

  labelC = IupLabel("Celsius =");
  labelF = IupLabel("Fahrenheit");

  tempC = IupText(NULL);
  IupSetAttribute(tempC, "SIZE", "50");
  IupSetAttribute(tempC, "NAME", "CELCIUS");
  IupSetAttribute(tempC, "MASK", IUP_MASK_FLOAT);
  tempF = IupText(NULL);
```

```
  IupSetAttribute(tempF, "SIZE", "50");
  IupSetAttribute(tempF, "NAME", "FAHRENHEIT");
  IupSetAttribute(tempF, "MASK", IUP_MASK_FLOAT);
  hbox = IupHbox(
    tempC,
    labelC,
    tempF,
    labelF,
    NULL);
  IupSetAttribute(hbox, "MARGIN", "10x10");
  IupSetAttribute(hbox, "GAP", "10");
  IupSetAttribute(hbox, "ALIGNMENT", "ACENTER");
  dlg = IupDialog(hbox);
  IupSetAttribute(dlg, "TITLE", "TempConv");

  IupSetAttribute(tempC, "VALUE", "");
  IupSetAttribute(tempF, "VALUE", "");

  /* Registers callbacks */
  IupSetCallback(tempC, "VALUECHANGED_CB", (Icallback)txt_celcius_cb);
  IupSetCallback(tempF, "VALUECHANGED_CB", (Icallback)txt_fahrenheit_cb);

  IupShowXY(dlg, IUP_CENTER, IUP_CENTER);

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")

function attrib2number(value)
  if not value or value == "" then
    return 0
  else
    return tonumber(value)
  end
end

function fahrenheit2celcius(temp)
```

```
  return (temp - 32) * (5./9.)
end

function celcius2fahrenheit(temp)
  return temp * (9./5.) + 32
end

--********************************** Main ****************************************

txt_celcius = iup.text{size = "60", mask = "[+/-]?(/d+/.?/d*|/./d+)"}
lbl_celcius = iup.label{title = "Celcius = "}

txt_fahrenheit = iup.text{size = "60", mask = "[+/-]?(/d+/.?/d*|/./d+)"}
lbl_fahrenheit = iup.label{title = "Fahrenheit"}

dlg = iup.dialog{iup.hbox{txt_celcius, lbl_celcius, txt_fahrenheit, lbl_fahrenheit;

dlg:showxy( iup.CENTER, iup.CENTER )

function txt_celcius:valuechanged_cb()
        txt_fahrenheit.value = celcius2fahrenheit(attrib2number(txt_celcius.value))
end

function txt_fahrenheit:valuechanged_cb()
        txt_celcius.value = fahrenheit2celcius(attrib2number(txt_fahrenheit.value))
end

if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
end
```

我们还可以在文本域失去焦点是更新它，通过 *KILLFOCUS_CB* 回调。

6.3 机票订购

挑战: 约束。

这个任务是构建一个窗口，窗口内部包含一个组合框 *C*，它有 *one-way flight* 和 *return flight* 两个选项。还有两个文本域 *T1* 和 *T2* 代表出发和返回日期。以及一个按钮 *B* 用来提交选择的航班。*C* 的值为 *return flight* 时,*T2* 才可用。*C* 的值为 *return flight*，并且 *T2* 的日期在 *T1* 之前，按钮 *B* 禁用。一个没有被禁用的文本域包含错误的日期格式，这个文本域就会被标红，并且按钮 *B* 会被禁用。我们点击按钮 *B* 后，就会有显示一条关于订购的消息[13]。开始时，*C* 的值为 *one-way flight*。*T1* 和 *T2* 含有一个相同的日期，这也意味着开始时 *T2* 是被禁用的。

[13]比如 *You have booked a one-way flight on 04.04.2014.*

C语言

```
#include <stdlib.h>
#include <iup.h>

int bt_book_cb(Ihandle *self)
{
  Ihandle* list = IupGetDialogChild(self, "LIST");
  Ihandle* startDate = IupGetDialogChild(self, "STARTDATE");
  Ihandle* endDate = IupGetDialogChild(self, "ENDDATE");

  int flightType = IupGetInt(list, "VALUE");

  if (flightType == 1)
    IupMessagef("Attention!", "You␣have␣booked␣a␣one-way␣flight␣on␣%s.", IupGetAttr
  else
    IupMessagef("Attention!", "You␣have␣booked␣a␣return␣flight␣on␣%s"
                              "␣and␣%s.", IupGetAttribute(startDate, "VALUE"), IupG

  return IUP_DEFAULT;
}

int validateDate(int day, int month, int year)
{
  int leapYear = 0;

  if (day < 1)
    return 1;

  if (year % 400 == 0 || year % 4 == 0 && year % 100 != 0)
    leapYear = 1;

  if (month == 2 && leapYear && day > 29)
    return 1;

  if (month == 2 && !leapYear && day > 28)
    return 1;

  if ((month == 4 || month == 6 || month == 9 || month == 11) && day > 30)
    return 1;

  if ((month == 1 || month == 3 || month == 5 || month == 7 ||
    month == 8 || month == 10 || month == 12) && day > 31)
```

```
    return 1;

  return 0;
}

int checkDate(Ihandle *self)
{
  int year, month, day;
  char *value = IupGetAttribute(self, "VALUE");
  char *part, *ret;
  int count = 1, notValid = 0;

  part = strtok(value, ".");
  if (part == NULL)
    return 1;
  while (part != NULL)
  {
    int len = strlen(part);
    switch (count)
    {
      case 1:
        if (strlen(part) > 2)
          notValid = 1;
        else
        {
          day = strtol(part, &ret, 10);
          if (*ret != '\0')
            notValid = 1;
        }
        break;
      case 2:
        if (strlen(part) > 2)
          notValid = 1;
        else
        {
          month = strtol(part, &ret, 10);
          if (*ret != '\0')
            notValid = 1;
        }
        break;
      case 3:
        if (strlen(part) != 4)
```

```
          notValid = 1;
        else
        {
          year = strtol(part, &ret, 10);
          if (*ret != '\0')
            notValid = 1;
        }
        break;
      default:
        notValid = 1;
    }
    count++;
    part = strtok(NULL, ".");
  }

  if (count < 4 || notValid)
    return 1;

  notValid = validateDate(day, month, year);

  if (notValid)
    IupSetAttribute(self, "VALID", "NO");
  else
  {
    IupSetAttribute(self, "VALID", "YES");
    IupSetInt(self, "DAY", day);
    IupSetInt(self, "MONTH", month);
    IupSetInt(self, "YEAR", year);
  }

  return notValid;
}

int startBeforeEnd(Ihandle *self)
{
  Ihandle* startDate = IupGetDialogChild(self, "STARTDATE");
  Ihandle* endDate = IupGetDialogChild(self, "ENDDATE");
  int startDay, startMonth, startYear;
  int endDay, endMonth, endYear;

  if (!IupGetInt(endDate, "ACTIVE"))
    return 1;
```

```
  startYear = IupGetInt(startDate, "YEAR");
  endYear = IupGetInt(endDate, "YEAR");

  if (startYear < endYear)
    return 1;

  startMonth = IupGetInt(startDate, "MONTH");
  endMonth = IupGetInt(endDate, "MONTH");

  if (startYear == endYear && startMonth < endMonth)
    return 1;

  startDay = IupGetInt(startDate, "DAY");
  endDay = IupGetInt(endDate, "DAY");

  if (startYear == endYear && startMonth == endMonth && startDay <= endDay)
    return 1;

  return 0;
}

int txt_valuechanged_cb(Ihandle *self)
{
  Ihandle *other = (Ihandle *)IupGetAttribute(self, "OTHER");
  Ihandle* button = IupGetDialogChild(self, "BUTTON");
  int notValid = checkDate(self);

  if (!notValid)
    IupSetAttribute(self, "BGCOLOR", "255 255 255");
  else
    IupSetAttribute(self, "BGCOLOR", "255 0 0");

  if (!notValid && (IupGetInt(other, "VALID") || !IupGetInt(other, "ACTIVE")) &&
      startBeforeEnd(self))
    IupSetAttribute(button, "ACTIVE", "YES");
  else
    IupSetAttribute(button, "ACTIVE", "NO");

  return IUP_DEFAULT;
}
```

```
int list_action_cb(Ihandle *self, char *text, int item, int state)
{
  Ihandle* startDate = IupGetDialogChild(self, "STARTDATE");
  Ihandle* endDate = IupGetDialogChild(self, "ENDDATE");
  Ihandle* button = IupGetDialogChild(self, "BUTTON");

  if (state == 0)
    return IUP_DEFAULT;

  if (item == 1)
    IupSetAttribute(endDate, "ACTIVE", "NO");
  else
    IupSetAttribute(endDate, "ACTIVE", "YES");

  if (item==2 && !startBeforeEnd(self))
      IupSetAttribute(button, "ACTIVE", "NO");

  return IUP_DEFAULT;
}

int main(int argc, char **argv)
{
  Ihandle *dlg, *vbox, *list, *startDate, *endDate, *button;

  IupOpen(&argc, &argv);

  list = IupList(NULL);

  IupSetAttribute(list, "NAME", "LIST");
  IupSetAttribute(list, "EXPAND", "HORIZONTAL");
  IupSetAttribute(list, "DROPDOWN", "YES");
  IupSetAttribute(list, "1", "one-way flight");
  IupSetAttribute(list, "2", "return flight");
  IupSetAttribute(list, "VALUE", "1");

  startDate = IupText(NULL);
  endDate = IupText(NULL);

  IupSetAttribute(startDate, "EXPAND", "HORIZONTAL");
  IupSetAttribute(startDate, "NAME", "STARTDATE");
  IupSetAttribute(startDate, "OTHER", (char *)endDate);
  IupSetAttribute(startDate, "VALUE", "22.09.1957");
```

```
  IupSetAttribute(startDate, "DAY", "22");
  IupSetAttribute(startDate, "MONTH", "09");
  IupSetAttribute(startDate, "YEAR", "1957");
  IupSetAttribute(startDate, "VALID", "YES");
  IupSetAttribute(endDate, "EXPAND", "HORIZONTAL");
  IupSetAttribute(endDate, "NAME", "ENDDATE");
  IupSetAttribute(endDate, "OTHER", (char *)startDate);
  IupSetAttribute(endDate, "VALUE", "22.09.1957");
  IupSetAttribute(endDate, "VALID", "YES");
  IupSetAttribute(endDate, "DAY", "22");
  IupSetAttribute(endDate, "MONTH", "09");
  IupSetAttribute(endDate, "YEAR", "1957");
  IupSetAttribute(endDate, "ACTIVE", "NO");

  button = IupButton("Book", NULL);

  IupSetAttribute(button, "EXPAND", "HORIZONTAL");
  IupSetAttribute(button, "NAME", "BUTTON");

  vbox = IupVbox(
    list,
    startDate,
    endDate,
    button,
    NULL);
  IupSetAttribute(vbox, "MARGIN", "10x10");
  IupSetAttribute(vbox, "GAP", "10");
  dlg = IupDialog(vbox);
  IupSetAttribute(dlg, "TITLE", "Book␣Flight");
  IupSetAttribute(dlg, "SIZE", "150");

  /* Registers callbacks */
  IupSetCallback(startDate, "VALUECHANGED_CB", (Icallback)txt_valuechanged_cb);
  IupSetCallback(endDate, "VALUECHANGED_CB", (Icallback)txt_valuechanged_cb);
  IupSetCallback(list, "ACTION", (Icallback)list_action_cb);
  IupSetCallback(button, "ACTION", (Icallback)bt_book_cb);

  IupShowXY(dlg, IUP_CENTER, IUP_CENTER);

  IupMainLoop();

  IupClose();
```

```
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")

function checkDateFormat(str_date)
        str_date = string.gsub(str_date, "^%s*(.-)%s*$", "%1")
        local check = string.find(str_date, "^%d?%d.%d?%d.%d%d%d%d$")
        if not check then
                return nil
        end
        local day, month, year = string.match(str_date, "(%d?%d).(%d?%d).(%d%d%d%d)
        if day==nil or month==nil or year==nil then
                return nil
        end
        return 1, tonumber(day), tonumber(month), tonumber(year)
end

function validateDate(day, month, year)
        local leapYear = 0

        if day and day < 1 or day>31 or month and month < 1 or month > 12 then
                return nil
        end

        if year % 400 == 0 or year % 4 == 0 and year % 100 ~= 0 then
                leapYear = 1
        end

        if month == 2 and leapYear and day > 29 then
                return nil
        end

        if month == 2 and not leapYear and day > 28 then
                return nil
        end

        if (month == 4 or month == 6 or month == 9 or month == 11) and day > 30 the
                return nil
        end
```

```
  return 1
end

function startBeforeEnd(startDay, startMonth, startYear, endDay, endMonth, endYear)

  if startYear < endYear then
    return 1
  end

  if  startYear == endYear and startMonth < endMonth then
    return 1
  end

  if  startYear == endYear and startMonth == endMonth and startDay <= endDay then
    return 1
  end

  return nil
end

--********************************** Main ****************************************

lst_flight = iup.list{"one-way flight", "return flight"; dropdown = "YES", value =
txt_startDate = iup.text{expand = "HORIZONTAL", value = "22.09.1957"}
txt_returnDate = iup.text{expand = "HORIZONTAL", active="NO", value = "22.09.1957"}
btn_book = iup.button{title = "Book", expand = "HORIZONTAL"}

dlg = iup.dialog{iup.vbox{lst_flight, txt_startDate, txt_returnDate, btn_book; gap=

function btn_book:action()
  local flightType = lst_flight.value
  if lst_flight.value == "1" then
    iup.Message("Atention!", "You have booked a one-way flight on "..txt_startDate.
  else
    iup.Message("Atention!", "You have booked a return flight on "..txt_startDate.v
                             " and "..txt_returnDate.value..".")
 end
end

function lst_flight:valuechanged_cb()
        if self.value == "1" then
                txt_returnDate.active = "NO"
```

```
        else
                txt_returnDate.active = "YES"
        end
end

function checkDate(self)
        local check, day, month, year = checkDateFormat(self.value)
        if not check then
                self.bgcolor = "255 0 0"
                self.valid = nil
                return
        end

        local valid = validateDate(day, month, year)
        if not valid then
                self.bgcolor = "255 0 0"
                self.valid = nil
                return
        end

        self.bgcolor = "255 255 255"
        self.valid = 1
        self.day = day
        self.month = month
        self.year = year
end

function txt_startDate:valuechanged_cb()
        checkDate(self)
        if self.valid and txt_returnDate.active == "NO" or
           (txt_returnDate.valid and startBeforeEnd(self.day, self.month, self.year
                                                    txt_returnDate.day, txt_retu
                btn_book.active = "YES"
        else
                btn_book.active = "NO"
        end
end

function txt_returnDate:valuechanged_cb()
        checkDate(self)
        if self.valid and txt_startDate.valid and
           startBeforeEnd(txt_startDate.day, txt_startDate.month, txt_startDate.yea
```

```
                                self.day, self.month, self.year) then
                btn_book.active = "YES"
        else
                btn_book.active = "NO"
        end
end

dlg:showxy( iup.CENTER, iup.CENTER )

if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
end
```

还可以使用 *IupDatePick* 控件来完成日期选择。

6.4 计时器

挑战: *并发，交互，反馈。*

在这个任务，我们构建一个窗口，里面包含了一个标尺 G，用来体现已经逝去的时间 e。一个标签用来显示 e，一个滑动条 S 用来调整计时时间 d。还有一个重置按钮 R。调整 S 会立即更新 d，而不是在松开鼠标之后才进行更新。移动 S，G 会立即被更新。当 $e >= d$ 计时器停止。之后，d 的增加，然后 $d > e$ 重新成立，计时器重新开始计时。单击按钮 R 重置 e 为 0。

C语言

```
#include <stdlib.h>
#include <iup.h>

int btn_reset_cb( Ihandle *self )
{
  Ihandle* text = IupGetDialogChild(self, "GAUGE");
  Ihandle* val = IupGetDialogChild(self, "VAL");
  Ihandle* dial = IupGetDialog(self);
  Ihandle* timer = (char *)IupGetAttribute(dial, "TIMER");

  IupSetAttribute(timer, "RUN", "NO");
  IupSetAttribute(timer, "RUN", "YES");

  IupSetAttribute(timer, "TOTALELAPSEDTIME", "0");
  IupSetAttribute(val, "VALUE", "30");

  return IUP_DEFAULT;
}

int val_valuechanged_cb(Ihandle *self)
```

```
{
  Ihandle* gauge = IupGetDialogChild(self, "GAUGE");
  Ihandle* dial = IupGetDialog(self);
  Ihandle* timer = (char *) IupGetAttribute(dial, "TIMER");
  double value = IupGetDouble(self, "VALUE");
  double totalTime = IupGetDouble(timer, "TOTALELAPSEDTIME");

  IupSetDouble(gauge, "MAX", value);

  if (totalTime/1000. < value)
    IupSetAttribute(timer, "RUN", "YES");
  else
    IupSetAttribute(gauge, "VALUE", IupGetAttribute(gauge, "MAX"));

  return IUP_DEFAULT;
}

int timer_cb(Ihandle *self)
{
  Ihandle* gauge = (Ihandle *)IupGetAttribute(self, "GAUGE");
  Ihandle* val = (Ihandle *)IupGetAttribute(self, "VAL");
  Ihandle* label = (Ihandle *)IupGetAttribute(self, "LABEL");
  double elapsedTime = IupGetDouble(self, "ELAPSEDTIME");
  double totaltime = IupGetDouble(self, "TOTALELAPSEDTIME");

  if ((totaltime + elapsedTime) / 1000. >= IupGetDouble(val, "VALUE"))
  {
    IupSetAttribute(self, "RUN", "NO");
    IupSetStrf(self, "TOTALELAPSEDTIME", "%.2lf", totaltime + elapsedTime);
    IupSetStrf(label, "TITLE", "%.2fs", IupGetDouble(val, "VALUE"));
    IupSetStrf(gauge, "VALUE", "%.2fs", IupGetDouble(val, "VALUE"));
  }
  else
  {
    IupSetStrf(label, "TITLE", "%.2fs", (totaltime + elapsedTime) / 1000.);
    IupSetStrf(gauge, "VALUE", "%.2fs", (totaltime + elapsedTime) / 1000.);
  }

  return IUP_DEFAULT;
}

int main(int argc, char **argv)
```

```
{
  Ihandle *dlg, *gauge, *label, *glabel, *val, *vlable, *button, *vbox, *hbox1, *hb

  IupOpen(&argc, &argv);

  timer = IupTimer();
  IupSetAttribute(timer, "TIME", "100");
  IupSetAttribute(timer, "TOTALELAPSEDTIME", "0");
  IupSetCallback(timer, "ACTION_CB", (Icallback)timer_cb);

  glabel = IupLabel("Elapsed Time:");
  gauge = IupGauge();
  IupSetAttribute(gauge, "NAME", "GAUGE");
  IupSetAttribute(gauge, "MIN", ".1");
  IupSetAttribute(gauge, "MAX", "30");
  IupSetAttribute(gauge, "VALUE", "30");
  IupSetAttribute(gauge, "EXPAND", "HORIZONTAL");
  IupSetAttribute(gauge, "SHOWTEXT", "NO");
  label = IupLabel("0.0s");
  IupSetAttribute(label, "NAME", "LABEL");
  IupSetAttribute(label, "EXPAND", "HORIZONTAL");
  vlable = IupLabel("Duration:");
  val = IupVal("HORIZONTAL");
  IupSetAttribute(val, "EXPAND", "HORIZONTAL");
  IupSetAttribute(val, "NAME", "VAL");
  IupSetAttribute(val, "MAX", "60.0");
  IupSetAttribute(val, "STEP", ".1");
  IupSetAttribute(val, "VALUE", "30.0");
  IupSetAttribute(val, "PAGESTEP", ".5");
  button = IupButton("Reset", NULL);
  IupSetAttribute(button, "EXPAND", "HORIZONTAL");
  IupSetAttribute(button, "NAME", "BUTTON");
  hbox1 = IupHbox(glabel, gauge, NULL);
  IupSetAttribute(hbox1, "MARGIN", "0x0");
  IupSetAttribute(hbox1, "ALIGNMENT", "ACENTER");
  hbox2 = IupHbox(vlable, val, NULL);
  IupSetAttribute(hbox2, "MARGIN", "0x0");
  IupSetAttribute(hbox2, "ALIGNMENT", "ACENTER");
  vbox = IupVbox(
    hbox1,
    label,
    hbox2,
```

```
    button,
    NULL);
  IupSetAttribute(vbox, "MARGIN", "10x10");
  IupSetAttribute(vbox, "GAP", "10");
  dlg = IupDialog(vbox);
  IupSetAttribute(dlg, "TITLE", "Timer");
  IupSetAttribute(dlg, "SIZE", "200");
  IupSetAttribute(dlg, "TIMER", (char *) timer);

  IupSetAttribute(timer, "GAUGE", (char *)gauge);
  IupSetAttribute(timer, "LABEL", (char *)label);
  IupSetAttribute(timer, "VAL", (char *)val);

  /* Registers callbacks */
  IupSetCallback(val, "VALUECHANGED_CB", (Icallback)val_valuechanged_cb);
  IupSetCallback(button, "ACTION", (Icallback)btn_reset_cb);

  IupShowXY(dlg, IUP_CENTER, IUP_CENTER);

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")

--********************************** Main *****************************************

timer = iup.timer{time = 100, run = "YES", totalelapsedtime = 0}
glabel = iup.label{title = "Elapsed Time:"}
gauge = iup.gauge{min = .1, max = 30, expand = "HORIZONATAL", showtext="NO"}
label = iup.label{text = "0.0s", expand = "HORIZONTAL"}
vlable = iup.label{title = "Duration:"}
val = iup.val{"HORIZONTAL"; expand = "HORIZONTAL", min = "0", max = "60", step = ".
button = iup.button{title = "Reset", expand = "HORIZONTAL"}

hbox1 = iup.hbox{glabel, gauge; margin = "0x0", alignment = "ACENTER"}
hbox2 = iup.hbox{vlable, val; margin = "0x0", alignment = "ACENTER"}
vbox = iup.vbox{hbox1, label, hbox2, button; margin = "10x10", gap = 10}
```

```
dlg = iup.dialog{vbox, title = "Timer", size = "200"}

function timer:action_cb()

  local elapsedTime = self.elapsedtime or 0
  local totaltime = self.totalelapsedtime

  local t = (totaltime + elapsedTime) / 1000.
  local valValue = val.value

  if t >= tonumber(valValue) then
    self.run = "NO"
    self.totalelapsedtime =  totaltime + elapsedTime
    label.title = string.format("%.2fs", val.value)
    gauge.value = val.value
  else
    label.title = string.format("%.2fs", (totaltime + elapsedTime) / 1000.)
    gauge.value = (totaltime + elapsedTime) / 1000.
        print(gauge.value.." - "..type(gauge.value))
  end
end

function val:valuechanged_cb()

  local totalTime = timer.totalelapsedtime

  gauge.max = self.value

  if totalTime/1000. < tonumber(self.value) then
    timer.run = "YES"
  else
    gauge.value = gauge.max
  end
end

function button:action()

  timer.run = "NO"

  timer.totalelapsedtime = 0
  val.value = "30."
  gauge.value = 0
```

```
  gauge.min = 0
  gauge.max = 60

  timer.run = "YES"
end

dlg:showxy( iup.CENTER, iup.CENTER )

if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
end
```

我们还可以用 *IupProgresssBar* 元素来代替 *IupGauge* 元素。本任务的一个加强就是只在单位度量的位置更新已经经过的时间。

6.5 增删改查

挑战: 区域划分，逻辑表现，管理变化，构建一个有价值的布局。

在这个任务，我们创建一个窗口包含下面的元素：一个文本域 *Tprefix*，一对文本域 *Tname* 和 *Tsurname*，一个列表框 *L*，按钮 *BC*，按钮 *BU* 和按钮 *BD*，三个标签。*L* 为数据提供了一个视图。*L* 是一个单选列表框。我们可以通过在 *Tprefix* 中输入文本来过滤列表框中的项目。单击按钮 *BC* 会组合 *Tname* 和 *Tsurname* 中的文本添加到 *L* 作为其中一个条目。按钮 *BU* 和按钮 *BD* 只有在 *L* 中有条目被选中时才被激活。与之相反，按钮 *BC* 和按钮 *BU* 不会添加条目到 *L*，它们会替换选中的条目的内容。*BD* 会删除选中的条目。窗口布局如图所示，特别的，*L* 必须占据窗口剩余的空间。

C语言

```
#include <stdlib.h>
#include <iup.h>

#define my_tolower(_c)  ((_c >= 'A' && _c <= 'Z')? (_c - 'A') + 'a': _c)

/********************************** Names List ************************************

#define MAX_NAMES 80

typedef struct _Name
{
  char name[80];
  char surname[80];
  int id;
} Name;

static Name namesList[MAX_NAMES];
static int namesCount = 0;
```

```
static int crud_create(char *surname, char *name)
{
  static int lastId = 0;

  if (namesCount == MAX_NAMES)
    return -1;

  strcpy(namesList[namesCount].surname, surname);
  strcpy(namesList[namesCount].name, name);
  namesList[namesCount].id = lastId;  /* each circle has an unique id */

  namesCount++;
  lastId++;
  return lastId - 1;
}

static Name crud_read(int index)
{
  return namesList[index];
}

static Name crud_find(int id)
{
  int i;
  for (i = 0; i < namesCount; i++)
  {
    if (namesList[i].id == id)
      break;
  }
  return namesList[i];
}

static void crud_update(int id, char *surname, char *name)
{
  int i;
  for (i = 0; i < namesCount; i++)
  {
    if (namesList[i].id == id)
      break;
  }
  strcpy(namesList[i].surname, surname);
```

```
  strcpy(namesList[i].name, name);
}

static void crud_delete(int id)
{
  int i, j;
  for (i = 0; i < namesCount; i++)
  {
    if (namesList[i].id == id)
      break;
  }

  for (j = i; j < namesCount - 1; j++)
    namesList[j] = namesList[j + 1];

  namesCount--;
}

static int crud_length()
{
  return namesCount;
}

/************************************ Operation List ****************************

void strLower(char* dstr, char* sstr)
{
  if (!sstr || sstr[0] == 0) return;
  for (; *sstr; sstr++, dstr++)
    *dstr = (char)my_tolower(*sstr);
  *dstr = 0;
}

static char *trim(char *str)
{
  char *last = NULL;
  while (*str==' ') str++;
  last = str + strlen(str);
  while (*--last == ' ');
  *(last + 1) = '\0';
  return str;
}
```

```
static int filterName(char *name, char *filter)
{
  char lname[80], lfilter[80];
  strLower(lname, name);
  strLower(lfilter, filter);
  int i = strncmp(lname, lfilter, strlen(filter));
  return (i==0);
}

static void loadList(Ihandle *ih)
{
  Ihandle *txtFilter = IupGetDialogChild(ih, "TXT_FILTER");
  Ihandle *listNames = IupGetDialogChild(ih, "LST_NAMES");
  Name data;
  int count = crud_length();
  char *filterTxt = IupGetAttribute(txtFilter, "VALUE");
  int filter = 1;
  int i, j;

  filterTxt = trim(filterTxt);
  if (*filterTxt == '\0')
    filter = 0;

  IupSetAttribute(listNames, "1", NULL);

  j = 1;
  for (i = 1; i <= count; i++)
  {
    Name name = crud_read(i-1);
    if (!filter || filterName(name.surname, filterTxt))
    {
      IupSetStrfId(listNames, "", j, "%s, %s", name.surname, name.name);
      IupSetIntId(listNames, "ID", j, name.id);
      j++;
    }
  }

  IupSetAttribute(listNames, "VALUE", NULL);
}

static void updateButtonsState(Ihandle *ih)
```

```
{
  Ihandle *updateButton = IupGetDialogChild(ih, "BTN_UPDATE");
  Ihandle *deleteButton = IupGetDialogChild(ih, "BTN_DELETE");
  Ihandle *listNames = IupGetDialogChild(ih, "LST_NAMES");

  if (!IupGetInt(listNames, "VALUE"))
  {
    IupSetAttribute(updateButton, "ACTIVE", "NO");
    IupSetAttribute(deleteButton, "ACTIVE", "NO");
  }
  else
  {
    IupSetAttribute(updateButton, "ACTIVE", "YES");
    IupSetAttribute(deleteButton, "ACTIVE", "YES");
  }
}

int btn_create_cb(Ihandle *ih)
{
  Ihandle *txtName = IupGetDialogChild(ih, "TXT_NAME");
  Ihandle *txtSurname = IupGetDialogChild(ih, "TXT_SURNAME");
  char *name = trim(IupGetAttribute(txtName, "VALUE"));
  char *surname = trim(IupGetAttribute(txtSurname, "VALUE"));
  crud_create(surname, name);
  IupSetAttribute(txtName, "VALUE", NULL);
  IupSetAttribute(txtSurname, "VALUE", NULL);
  loadList(ih);
  updateButtonsState(ih);
  return IUP_DEFAULT;
}

int btn_update_cb(Ihandle *ih)
{
  Ihandle *txtName = IupGetDialogChild(ih, "TXT_NAME");
  Ihandle *txtSurname = IupGetDialogChild(ih, "TXT_SURNAME");
  Ihandle *listNames = IupGetDialogChild(ih, "LST_NAMES");
  int index = IupGetInt(listNames, "VALUE");
  int id = IupGetIntId(listNames, "ID", index);
  char *name = trim(IupGetAttribute(txtName, "VALUE"));
  char *surname = trim(IupGetAttribute(txtSurname, "VALUE"));
  crud_update(id, surname, name);
  IupSetAttribute(txtName, "VALUE", NULL);
```

```
  IupSetAttribute(txtSurname, "VALUE", NULL);
  loadList(ih);
  updateButtonsState(ih);
  return IUP_DEFAULT;
}

int btn_delete_cb(Ihandle *ih)
{
  Ihandle *txtName = IupGetDialogChild(ih, "TXT_NAME");
  Ihandle *txtSurname = IupGetDialogChild(ih, "TXT_SURNAME");
  Ihandle *listNames = IupGetDialogChild(ih, "LST_NAMES");
  int index = IupGetAttribute(listNames, "VALUE");
  int id = IupGetIntId(listNames, "ID", index);
  char *name = trim(IupGetAttribute(txtName, "VALUE"));
  char *surname = trim(IupGetAttribute(txtSurname, "VALUE"));
  crud_delete(id, surname, name);
  IupSetAttribute(txtName, "VALUE", NULL);
  IupSetAttribute(txtSurname, "VALUE", NULL);
  loadList(ih);
  updateButtonsState(ih);
  return IUP_DEFAULT;
}

int lst_valuechanged_cb(Ihandle *ih)
{
  Ihandle *txtName = IupGetDialogChild(ih, "TXT_NAME");
  Ihandle *txtSurname = IupGetDialogChild(ih, "TXT_SURNAME");
  int sel = IupGetInt(ih, "VALUE");
  int id = IupGetIntId(ih, "ID", sel);
  Name name = crud_find(id);
  IupSetAttribute(txtName, "VALUE", name.name);
  IupSetAttribute(txtSurname, "VALUE", name.surname);
  updateButtonsState(ih);
  return IUP_DEFAULT;
}

int txt_valuechanged_cb(Ihandle *ih)
{
  loadList(ih);
  return IUP_DEFAULT;
}
```

```
int main(int argc, char **argv)
{
  Ihandle *dlg, *lbl_filter, *txt_filter, *lst_names, *lbl_name, *txt_name, *lbl_sur
  Ihandle *btn_create, *btn_update, *btn_delete;
  Ihandle *hbox, *box, *hbx_filter, *hbx_top, *hbx_name, *hbx_surname, *hbx_buttons

  IupOpen(&argc, &argv);

  lbl_filter = IupLabel("Filter prefix:");
  IupSetAttribute(lbl_filter, "NAME", "LBL_FILTER");
  txt_filter = IupText(NULL);
  IupSetAttribute(txt_filter, "NAME", "TXT_FILTER");
  IupSetAttribute(txt_filter, "EXPAND", "HORIZONTAL");
  IupSetAttribute(txt_filter, "SIZE", "60x");
  lst_names = IupList(NULL);
  IupSetAttribute(lst_names, "NAME", "LST_NAMES");
  IupSetAttribute(lst_names, "EXPAND", "YES");
  IupSetAttribute(lst_names, "SIZE", "60x");
  IupSetAttribute(lst_names, "VISIBLELINES", "6");
  lbl_name = IupLabel("Name:");
  IupSetAttribute(lbl_name, "NAME", "LBL_NAME");
  IupSetAttribute(lbl_name, "SIZE", "35");
  txt_name = IupText(NULL);
  IupSetAttribute(txt_name, "NAME", "TXT_NAME");
  IupSetAttribute(txt_name, "SIZE", "60");
  IupSetAttribute(txt_name, "NC", "80");
  IupSetAttribute(txt_name, "EXPAND", "HORIZONTAL");
  lbl_surname = IupLabel("Surname:");
  IupSetAttribute(lbl_surname, "NAME", "LBL_SURNAME");
  IupSetAttribute(lbl_surname, "SIZE", "35");
  txt_surname = IupText(NULL);
  IupSetAttribute(txt_surname, "NAME", "TXT_SURNAME");
  IupSetAttribute(txt_surname, "SIZE", "60");
  IupSetAttribute(txt_surname, "NC", "80");
  IupSetAttribute(txt_surname, "EXPAND", "HORIZONTAL");

  btn_create = IupButton("Create", NULL);
  IupSetAttribute(btn_create, "NAME", "BTN_CREATE");
  IupSetAttribute(btn_create, "SIZE", "30");
  btn_update = IupButton("Update", NULL);
  IupSetAttribute(btn_update, "NAME", "BTN_UPDATE");
  IupSetAttribute(btn_update, "SIZE", "30");
```

```
  IupSetAttribute(btn_update, "ACTIVE", "NO");
  btn_delete = IupButton("Delete", NULL);
  IupSetAttribute(btn_delete, "NAME", "BTN_DELETE");
  IupSetAttribute(btn_delete, "SIZE", "30");
  IupSetAttribute(btn_delete, "ACTIVE", "NO");

  hbx_filter = IupHbox(lbl_filter, txt_filter, NULL);
  IupSetAttribute(hbx_filter, "ALIGNMENT", "ACENTER");
  hbx_top = IupHbox(hbx_filter, IupFill(), NULL);
  IupSetAttribute(hbx_top, "HOMOGENEOUS", "YES");
  hbx_name = IupHbox(lbl_name, txt_name, NULL);
  IupSetAttribute(hbx_name, "ALIGNMENT", "ACENTER");
  hbx_surname = IupHbox(lbl_surname, txt_surname, NULL);
  IupSetAttribute(hbx_surname, "alignment", "ACENTER");
  hbx_buttons = IupHbox(btn_create, btn_update, btn_delete, NULL);

  vbx_input = IupVbox(hbx_name, hbx_surname, NULL);

  hbox = IupHbox(lst_names, vbx_input, NULL);
  IupSetAttribute(hbox, "HOMOGENEOUS", "YES");
  box = IupVbox(hbx_top, hbox, hbx_buttons, NULL);
  IupSetAttribute(box, "NMARGIN", "10x10");

  dlg = IupDialog(box);
  IupSetAttribute(dlg, "TITLE", "CRUD");
  IupSetAttribute(dlg, "GAP", "10");

  /* Registers callbacks */
  IupSetCallback(btn_create, "ACTION", (Icallback)btn_create_cb);
  IupSetCallback(btn_update, "ACTION", (Icallback)btn_update_cb);
  IupSetCallback(btn_delete, "ACTION", (Icallback)btn_delete_cb);
  IupSetCallback(lst_names, "VALUECHANGED_CB", (Icallback)lst_valuechanged_cb);
  IupSetCallback(txt_filter, "VALUECHANGED_CB", (Icallback)txt_valuechanged_cb);

  IupShowXY(dlg, IUP_CENTER, IUP_CENTER);

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")

data = {}
lastId = 0

function crud_create(surname, name)
        local item = {}
        lastId = lastId + 1
        item.name = name
        item.surname = surname
        item.id = lastId
        data[item.id] = item
        return lastId
end

function crud_read(nextId)
        local n, v = next(data, nextId)
        if not n then
                return n
        end
        return n, v.surname, v.name
end

function crud_update(id, surname, name)
        data[id].name = name
        data[id].surname = surname
end

function crud_delete(id)
        data[id] = nil
end

--********************************** Main ****************************************

lbl_filter = iup.label{title = "Filter prefix:"}
txt_filter = iup.text{size = "60x", expand = "HORIZONTAL"}
lst_names = iup.list{size = "60x", expand = "YES", visiblelines=6}
lbl_name = iup.label{title = "Name:", size = 35}
txt_name = iup.text{size = 60, expand = "HORIZONTAL"}
lbl_surname = iup.label{title = "Surname:", size = 35}
txt_surname = iup.text{size = 60, expand = "HORIZONTAL"}
```

```
btn_create = iup.button{title = "Create", size = "30"}
btn_update = iup.button{title = "Update", size = "30", active = "NO"}
btn_delete = iup.button{title = "Delete", size = "30", active = "NO"}

hbx_filter = iup.hbox{iup.hbox{lbl_filter, txt_filter; alignment = "ACENTER"}, iup.
hbx_name = iup.hbox{lbl_name, txt_name; alignment = "ACENTER"}
hbx_surname = iup.hbox{lbl_surname, txt_surname; alignment = "ACENTER"}
hbx_buttons = iup.hbox{btn_create, btn_update, btn_delete}

vbx_input = iup.vbox{hbx_name, hbx_surname}

hbox = iup.hbox{lst_names, vbx_input; homogeneous = "YES"}
box = iup.vbox{hbx_filter, hbox, hbx_buttons; nmargin = "10x10"}

dlg = iup.dialog{box, title = "CRUD", gap = "10"}

function trim (str)
return string.gsub(str, "^%s*(.-)%s*$", "%1")
end

function filterName(name, filter)
        local i,j = string.find(string.lower(name), string.lower(filter))
        return i==1
end

function loadList()
        local currId = nil
        local surname, name
        local count = 1
        local filter = txt_filter.value
        if filter then
                filter = trim(filter)
                if filter == "" then
                        filter = nil
                end
        end
        lst_names[1] = nil
        currId, surname, name = crud_read(currId)
        while currId do
                if not filter or filterName(surname, filter) then
                        lst_names[count] = surname.. ", ".. name
```

```
                        lst_names["ID"..count] = currId
                        count = count + 1
                end
                        currId, surname, name = crud_read(currId)
        end
        lst_names.value = 0
end

function updateButtonsState()
        if lst_names.value == "0" then
                btn_delete.active = "NO"
                btn_update.active = "NO"
        else
                btn_delete.active = "YES"
                btn_update.active = "YES"
        end
end

function lst_names:valuechanged_cb()
        updateButtonsState()
end

function txt_filter:valuechanged_cb()
        loadList(self.value)
end

function btn_create:action()
        local name = trim(txt_name.value)
        local surname = trim(txt_surname.value)
        crud_create(surname, name)
        txt_name.value = ""
        txt_surname.value = ""
        loadList()
        updateButtonsState()
end

function btn_update:action()
        local index = lst_names.value
        local id = lst_names["ID"..index]
        local name = trim(txt_name.value)
        local surname = trim(txt_surname.value)
        crud_update(tonumber(id), surname, name)
```

```
        txt_name.value = ""
        txt_surname.value = ""
        loadList()
        updateButtonsState()
end

function btn_delete:action()
        local index = lst_names.value
        local id = lst_names["ID"..index]
        crud_delete(tonumber(id), surname, name)
        txt_name.value = ""
        txt_surname.value = ""
        loadList()
        updateButtonsState()
end

dlg:showxy( iup.CENTER, iup.CENTER )

if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
end
```

6.6 画圆

挑战: 撤销/重做，自定义绘制，对话框控制。

在这个任务，我们构建一个窗口，它包含了撤销和重做按钮和一个画布。点击鼠标左键会绘制一个没有填充的圆形。在一个圆形中绘制左键，会给圆形填充灰色，表示选中了这个圆形。点击鼠标右键会弹出一个菜单，这个菜单包含了一个 *Adjust diameter..* 菜单项目。这个菜单项目允许我们调整圆形的直径大小。更改圆形大小后，圆形原来的直径大小会被保存用作撤销/重做功能。点击 *undo* 按钮会撤回最近的有意义的操作，点击 *redo* 按钮会恢复最近撤销的操作。

C语言

```
#include <stdlib.h>
#include <math.h>
#include <iup.h>
#include <iupdraw.h>


/********************************** Circle List *****************************

#define DEF_RADIUS 30
#define MAX_RADIUS 100
#define MAX_CIRCLES 80
```

```
typedef struct _Circle
{
  int x;
  int y;
  int r;
  int id;
} Circle;

static Circle circlesList[MAX_CIRCLES];
static int circlesCount = 0;

static int circles_add(int x, int y)
{
  static int lastId = 0;

  if (circlesCount == MAX_CIRCLES)
    return -1;

  circlesList[circlesCount].x = x;
  circlesList[circlesCount].y = y;
  circlesList[circlesCount].r = DEF_RADIUS;
  circlesList[circlesCount].id = lastId;  /* each circle has an unique id */

  circlesCount++;
  lastId++;
  return lastId - 1;
}

static int circles_insert(Circle circle)
{
  if (circlesCount == MAX_CIRCLES)
    return -1;

  circlesList[circlesCount] = circle;
  circlesCount++;
  return 1;
}

static void circles_remove(Circle circle)
{
  int i, j;
```

```
  for (i = 0; i < circlesCount; i++)
  {
    if (circlesList[i].id == circle.id)
      break;
  }

  for (j = i; j < circlesCount - 1; j++)
    circlesList[j] = circlesList[j + 1];

  circlesCount--;
}

static void circles_update(Circle circle)
{
  int i;
  for (i = 0; i < circlesCount; i++)
  {
    if (circlesList[i].id == circle.id)
      break;
  }

  circlesList[i].r = circle.r;
}

static int circles_pick(int x, int y)
{
  int i;
  int min_r = MAX_RADIUS;
  int id = -1;

  for (i = 0; i < circlesCount; i++)
  {
    int dx = circlesList[i].x - x;
    int dy = circlesList[i].y - y;
    int r = dx*dx + dy*dy;

    r = (int)sqrt((double)r);

    if (r < circlesList[i].r)
    {
      if (id < 0)
      {
```

```
        id = circlesList[i].id;
        min_r = r;
      }
      else if (r < min_r)
        id = circlesList[i].id;
    }
  }

  return id;
}

static Circle circles_find(int id)
{
  int i;
  for (i = 0; i < circlesCount; i++)
  {
    if (circlesList[i].id == id)
      break;
  }
  return circlesList[i];
}


/************************************ Operation List ****************************

#define MAX_OPERATIONS 200

enum { OP_CREATE, OP_REMOVE, OP_UPDATE };

typedef struct _Operation
{
  int type;
  Circle circle;
} Operation;

static Operation operationsList[MAX_OPERATIONS];
static int operationsCount = 0;
static int currentOperation = -1;

static void op_add(int type, Circle circle)
{
  if (operationsCount == MAX_OPERATIONS)
```

```
    return;

  if (currentOperation < operationsCount - 1)
    operationsCount = currentOperation + 1;

  operationsList[operationsCount].type = type;
  operationsList[operationsCount].circle = circle;
  currentOperation = operationsCount;

  operationsCount++;
}

static void op_undo(void)
{
  Operation op;

  if (currentOperation < 0)
    return;

  op = operationsList[currentOperation];
  currentOperation--;

  switch (op.type)
  {
    case OP_CREATE:
      circles_remove(op.circle);
      break;
    case OP_REMOVE:
      circles_insert(op.circle);
      break;
    case OP_UPDATE:
      op = operationsList[currentOperation];
      currentOperation--; /* return 2 to get the old value */

      circles_update(op.circle);
      break;
  }
}

static void op_redo(void)
{
  Operation op;
```

```
  if (currentOperation + 1 == operationsCount)
    return;

  currentOperation++;
  op = operationsList[currentOperation];

  switch (op.type)
  {
    case OP_CREATE:
      circles_insert(op.circle);
      break;
    case OP_REMOVE:
      circles_remove(op.circle);
      break;
    case OP_UPDATE:
      currentOperation++; /* increment 2 to get the new value */
      op = operationsList[currentOperation];

      circles_update(op.circle);
      break;
  }
}

static int op_has_undo(void)
{
  if (currentOperation < 0)
    return 0;

  return 1;
}

static int op_has_redo(void)
{
  if (currentOperation + 1 == operationsCount)
    return 0;

  return 1;
}


/****************************************** Interface **********************
```

```
void update_buttons(Ihandle* ih)
{
  Ihandle *undoButton = IupGetDialogChild(ih, "UNDO");
  Ihandle *redoButton = IupGetDialogChild(ih, "REDO");

  if (op_has_undo())
    IupSetAttribute(undoButton, "ACTIVE", "YES");
  else
    IupSetAttribute(undoButton, "ACTIVE", "NO");

  if (op_has_redo())
    IupSetAttribute(redoButton, "ACTIVE", "YES");
  else
    IupSetAttribute(redoButton, "ACTIVE", "NO");
}

int bt_undo_cb(Ihandle *self)
{
  Ihandle *canvas = IupGetDialogChild(self, "CANVAS");

  op_undo();

  update_buttons(self);

  IupUpdate(canvas);

  return IUP_DEFAULT;
}

int bt_redo_cb(Ihandle *self)
{
  Ihandle *canvas = IupGetDialogChild(self, "CANVAS");

  op_redo();

  update_buttons(self);

  IupUpdate(canvas);

  return IUP_DEFAULT;
}
```

```
static int canvas_button_cb(Ihandle *ih, int but, int pressed, int x, int y, char*
{
  Ihandle *dlg = IupGetDialog(ih);

  if (but == IUP_BUTTON1 && pressed)
  {
    int id = circles_pick(x, y);
    if (id >= 0)
      IupSetInt(ih, "HIGHLIGHTEDCIRCLE", id);
    else
    {
      id = circles_add(x, y);
      op_add(OP_CREATE, circles_find(id));

      update_buttons(ih);
    }

    IupUpdate(ih);
  }
  else if (but == IUP_BUTTON3 && pressed)
  {
    int highlightedCircle = IupGetInt(ih, "HIGHLIGHTEDCIRCLE");
    int id = circles_pick(x, y);
    if (id == highlightedCircle)
    {
      Circle circle = circles_find(id);
      int new_r, old_r = circle.r;
      Ihandle *configDialog = (Ihandle *)IupGetAttribute(dlg, "CONFIGDIALOG");
      Ihandle *val = IupGetDialogChild(configDialog, "VAL");
      Ihandle *lbl = IupGetDialogChild(configDialog, "LBL");

      IupSetStrf(lbl, "TITLE", "Adjust␣the␣diameter␣of␣the␣circle␣at␣(%d,␣%d)", cir
      IupSetInt(val, "VALUE", old_r);
      IupSetInt(configDialog, "CIRCLEID", id);

      IupPopup(configDialog, IUP_CENTERPARENT, IUP_CENTERPARENT);

      new_r = IupGetInt(val, "VALUE");
      if (new_r != old_r)
      {
        op_add(OP_UPDATE, circle); /* add both old and new */
```

```
        circle.r = new_r;
        op_add(OP_UPDATE, circle);

        update_buttons(ih);
      }
    }
  }

  (void)status;
  return IUP_DEFAULT;
}

static int canvas_action(Ihandle *ih)
{
  int i, w, h, highlightedCircle;

  IupDrawBegin(ih);

  IupDrawGetSize(ih, &w, &h);

  IupSetAttribute(ih, "DRAWCOLOR", "255 255 255");
  IupSetAttribute(ih, "DRAWSTYLE", "FILL");
  IupDrawRectangle(ih, 0, 0, w - 1, h - 1);

  IupSetAttribute(ih, "DRAWCOLOR", "0 0 0");
  IupSetAttribute(ih, "DRAWSTYLE", "STROKE");

  highlightedCircle = IupGetInt(ih, "HIGHLIGHTEDCIRCLE");

  for (i = 0; i < circlesCount; i++)
  {
    Circle circle = circlesList[i];

    if (highlightedCircle == circle.id)
    {
      IupSetAttribute(ih, "DRAWCOLOR", "128 128 128");
      IupSetAttribute(ih, "DRAWSTYLE", "FILL");
    }
    else
    {
      IupSetAttribute(ih, "DRAWCOLOR", "0 0 0");
      IupSetAttribute(ih, "DRAWSTYLE", "STROKE");
```

```
    }

    IupDrawArc(ih, circle.x - circle.r, circle.y - circle.r,
                   circle.x + circle.r, circle.y + circle.r, 0., 360.);
  }

  IupDrawEnd(ih);
  return IUP_DEFAULT;
}

int val_valuechanged_cb(Ihandle *self)
{
  Ihandle* dlg = IupGetDialog(self);
  Ihandle* canvas = (Ihandle *)IupGetAttribute(self, "CANVAS");
  int circleId = IupGetInt(dlg, "CIRCLEID");
  int r = IupGetInt(self, "VALUE");
  Circle circle = circles_find(circleId);

  circle.r = r;
  circles_update(circle);

  IupUpdate(canvas);

  return IUP_DEFAULT;
}

Ihandle *createCircleConfigDialog(Ihandle* parent_dlg)
{
  Ihandle *dlg, *val, *lbl, *box;

  lbl = IupLabel(NULL);
  IupSetAttribute(lbl, "EXPAND", "HORIZONTAL");
  IupSetAttribute(lbl, "TITLE", "Adjust␣the␣diameter␣of␣the␣circle␣at␣(100,␣100)");
  IupSetAttribute(lbl, "NAME", "LBL");

  val = IupVal("HORIZONTAL");
  IupSetAttribute(val, "EXPAND", "HORIZONTAL");
  IupSetAttribute(val, "MIN", "0");
  IupSetInt(val, "MAX", MAX_RADIUS);
  IupSetAttribute(val, "NAME", "VAL");

  box = IupVbox(IupFill(), lbl, val, IupFill(), NULL);
```

```
  IupSetAttribute(box, "ALIGMENT", "ACENTER");

  IupSetAttribute(box, "NMARGIN", "10x10");
  dlg = IupDialog(box);
  IupSetAttribute(dlg, "TITLE", "Circle Config");
  IupSetAttributeHandle(dlg, "PARENTDIALOG", parent_dlg);

  IupSetCallback(val, "VALUECHANGED_CB", (Icallback)val_valuechanged_cb);

  return dlg;
}

int main(int argc, char **argv)
{
  Ihandle *dlg, *hbox, *vbox, *configDialog;
  Ihandle *undoButton, *redoButton, *canvas;

  IupOpen(&argc, &argv);

  undoButton = IupButton("Undo", NULL);
  IupSetAttribute(undoButton, "NAME", "UNDO");
  IupSetAttribute(undoButton, "SIZE", "60");
  redoButton = IupButton("Redo", NULL);
  IupSetAttribute(redoButton, "NAME", "REDO");
  IupSetAttribute(redoButton, "SIZE", "60");

  IupSetAttribute(redoButton, "ACTIVE", "NO");
  IupSetAttribute(undoButton, "ACTIVE", "NO");

  canvas = IupCanvas(NULL);
  IupSetAttribute(canvas, "NAME", "CANVAS");
  IupSetAttribute(canvas, "EXPAND", "YES");
  IupSetInt(canvas, "HIGHLIGHTEDCIRCLE", -1);

  hbox = IupHbox(IupFill(), undoButton, redoButton, IupFill(), NULL);

  vbox = IupVbox(hbox, canvas, NULL);

  IupSetAttribute(vbox, "NMARGIN", "10x10");
  IupSetAttribute(vbox, "GAP", "10");
  dlg = IupDialog(vbox);
  IupSetAttribute(dlg, "TITLE", "Circle Drawer");
```

```
  IupSetAttribute(dlg, "SIZE", "300x150");

  /* Registers callbacks */
  IupSetCallback(undoButton, "ACTION", (Icallback)bt_undo_cb);
  IupSetCallback(redoButton, "ACTION", (Icallback)bt_redo_cb);
  IupSetCallback(canvas, "ACTION", (Icallback)canvas_action);
  IupSetCallback(canvas, "BUTTON_CB", (Icallback)canvas_button_cb);

  configDialog = createCircleConfigDialog(dlg);

  IupSetAttribute(configDialog, "CANVAS", (char *)canvas);

  IupSetAttribute(dlg, "CONFIGDIALOG", (char *)configDialog);

  IupShowXY(dlg, IUP_CENTER, IUP_CENTER);

  IupMainLoop();

  IupClose();
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")


--********************************** Circle List *********************************

DEF_RADIUS = 30
MAX_RADIUS = 100
circlesList = {}
circlesCount = 0
lastId = 1

function circles_add(x, y)
        circlesCount = circlesCount + 1
        circlesList[circlesCount] = {}
        circlesList[circlesCount].x = x
        circlesList[circlesCount].y = y
        circlesList[circlesCount].r = DEF_RADIUS
        circlesList[circlesCount].id = lastId  -- each circle has an unique id
        lastId = lastId + 1
```

```
        return lastId - 1
end

function circles_insert(id, x, y, r)
        circlesCount = circlesCount + 1
        circlesList[circlesCount] = {}
        circlesList[circlesCount].x = x
        circlesList[circlesCount].y = y
        circlesList[circlesCount].r = r
        circlesList[circlesCount].id = id
        return 1
end

function circles_remove(id)
        local index
        for i = 1, circlesCount do
                if circlesList[i].id == id then
                        index = i
                        break
                end
        end
        for j = index, circlesCount - 1 do
                circlesList[j] = circlesList[j + 1]
        end
        circlesCount = circlesCount - 1
end

function circles_update(id, r)
        local index
        for i = 1, circlesCount do
                if circlesList[i].id == id then
                        index = i
                        break
                end
        end
        circlesList[index].r = r
end

function circles_pick(x, y)
        local min_r = MAX_RADIUS
        local id = -1
        for i = 1, circlesCount do
```

```
                local dx = circlesList[i].x - x
                local dy = circlesList[i].y - y
                local r = dx*dx + dy*dy
                r = r^0.5
                if r < circlesList[i].r then
                        if id < 0 then
                                id = circlesList[i].id
                                min_r = r
                        elseif r < min_r then
                                id = circlesList[i].id
                        end
                end
        end
        return id
end

function circles_find(id)
        local index
        for i = 1, circlesCount do
                if circlesList[i].id == id then
                        index = i
                        break
                end
        end
        return circlesList[index]
end

--********************************** Operation List ***************************

MAX_OPERATIONS = 200

OP_CREATE = 1
OP_REMOVE = 2
OP_UPDATE = 3

operationsList = {}
operationsCount = 0
currentOperation = 0

function op_add(op_type, id, x, y, r)
        if currentOperation < operationsCount then
                operationsCount = currentOperation
```

```
        end
        operationsCount = operationsCount + 1
        operationsList[operationsCount] = {}
        operationsList[operationsCount].type = op_type
        operationsList[operationsCount].x = x
        operationsList[operationsCount].y = y
        operationsList[operationsCount].r = r
        operationsList[operationsCount].id = id
        currentOperation = operationsCount
end

function op_undo(void)
        if currentOperation < 1 then
                return
        end

        local op = operationsList[currentOperation]
        currentOperation = currentOperation - 1

        if op.type == OP_CREATE then
                circles_remove(op.id)
        elseif op.type == OP_REMOVE then
                circles_insert(op.id, op.x, op.y, op.r)
        elseif op.type == OP_UPDATE then
                op = operationsList[currentOperation]
                currentOperation = currentOperation - 1 -- return 2 to get the old

                circles_update(op.id, op.r)
        end
end

function op_redo(void)
        if currentOperation == operationsCount then
                return
        end

        currentOperation = currentOperation + 1
        op = operationsList[currentOperation]

        if op.type == OP_CREATE then
                circles_insert(op.id, op.x, op.y, op.r)
        elseif op.type == OP_REMOVE then
```

```
                circles_remove(op.id)
        elseif op.type == OP_UPDATE then
                currentOperation = currentOperation + 1 -- increment 2 to get the n
                op = operationsList[currentOperation]

                circles_update(op.id, op.r)
        end
end

function op_has_undo()
        if currentOperation < 1 then
                return nil
        end
        return 1
end

function op_has_redo()
        if currentOperation == operationsCount then
                return nil
        end
        return 1
end


--------------------------------------------- Interface ----------------------

function update_buttons()
        if op_has_undo() then
                undoButton.active = "YES"
          else
                undoButton.active = "NO"
        end

        if op_has_redo() then
                redoButton.active = "YES"
          else
                redoButton.active = "NO"
        end
end

undoButton = iup.button{title = "Undo", size = "60", active = "NO"}
redoButton = iup.button{title = "Redo", size = "60", active = "NO"}
```

```
canvas = iup.canvas{expand = "YES", highlightedCircle = -1}

hbox = iup.hbox{iup.fill{}, undoButton, redoButton, iup.fill{}}

vbox = iup.vbox{hbox, canvas;  nmargin = "10x10", gap = "10"}

dlg = iup.dialog{vbox, title = "Circle Drawer", size = "300x150"}

val_lbl = iup.label{"Adjust the diameter of the circle at (100, 100)"; expand = "HO

val = iup.val{"HORIZONTAL"; expand = "HORIZONTAL", min = "0", max = MAX_RADIUS}

config_box = iup.vbox{iup.fill{}, val_lbl, val, iup.fill{}, alignment = "ACENTER",

config_dial = iup.dialog{config_box; parentdialog = dlg, title = "Circle Config"}

function undoButton:action()
        op_undo()
        update_buttons()
        iup.Update(canvas)
end

function redoButton:action()
        op_redo()
        update_buttons()
        iup.Update(canvas)
end

function canvas:action(posx, posy)
        iup.DrawBegin(self)
        local w, h = iup.DrawGetSize(self)
        self.drawcolor = "255 255 255"
        self.drawstyle = "FILL"
        iup.DrawRectangle(self, 0, 0, w - 1, h - 1)
        self.drawcolor = "0 0 0"
        self.drawstyle = "STROKE"
        local highlightedCircle = tonumber(self.highlightedcircle)
        for i = 1, circlesCount do
    local circle = circlesList[i]
                if highlightedCircle == circle.id then
                        self.drawcolor = "128 128 128"
```

```
                        self.drawstyle = "FILL"
                else
                        self.drawcolor = "0 0 0"
                        self.drawstyle = "STROKE"
                end
                iup.DrawArc(self, circle.x - circle.r, circle.y - circle.r,
                      circle.x + circle.r, circle.y + circle.r, 0., 360.)
        end
        iup.DrawEnd(self)
end

function canvas:button_cb(button, pressed, x, y, status)
        if button == iup.BUTTON1 and pressed == 1 then
                local id = circles_pick(x, y)
                if id > 0 then
                        self.highlightedcircle = id
                else
                        id = circles_add(x, y)
                        local circle = circles_find(id)
                        op_add(OP_CREATE, circle.id, circle.x, circle.y, circle.r)
                        update_buttons()
                end
                iup.Update(self)
        elseif button == iup.BUTTON3 and pressed == 1 then
                local highlightedCircle = tonumber(self.highlightedcircle)
                local id = circles_pick(x, y)
                if id == highlightedCircle then
                        local circle = circles_find(id)
                        local old_r = circle.r
                        local new_r = old_r
                        val_lbl.title = string.format("Adjust the diameter of the c
                        val.value = old_r
                        config_dial.circle_id = id
                        iup.Popup(config_dial, iup.CENTERPARENT, iup.CENTERPARENT)
                        new_r = tonumber(val.value)
                        if new_r ~= old_r then
                                op_add(OP_UPDATE, circle.id, circle.x, circle.y, old
                                op_add(OP_UPDATE, circle.id, circle.x, circle.y, new
                                update_buttons()
                        end
                end
        end
```

```
end

function val:valuechanged_cb()
        local circle_id = tonumber(config_dial.circle_id)
        local r = tonumber(self.value)
        circles_update(circle_id, r)
        iup.Update(canvas)
end

dlg:showxy( iup.CENTER, iup.CENTER )

if (iup.MainLoopLevel()==0) then
        iup.MainLoop()
end
```

撤销和重做都是我们自己手动实现的。本例有许多可以加强的地方，特别是对圆形进行处理的地方。

6.7 表单

挑战: 改变传播，组建定制，实现一个复杂界面的程序。

在这个任务中，我们创建一个简单但非常有用的表格程序。这个表格是带有滚动条的。表格的行范围是 0 到 99，列范围是 A 到 Z。双击一个单元格，就可以编辑这个单元格的公式。结束后，公式的计算结果就会显示在单元格中。此外，依赖这个单元格的其它单元格也会被更新，也就是说这个更新是连锁反应的。如果存在一个表格控件，就不应该使用它。我们可以使用相似的控件来定制一个可重用的表格组件。

C语言

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

#include <iup.h>
#include <iupcontrols.h>

enum { OP_SUM };

typedef struct _Cell
{
  int lin;
  int col;
} Cell;

typedef struct _Operation
```

```
{
  int type;
  int startLin;
  int endLin;
  int startCol;
  int endCol;
} Operation;

enum
{
  FORM_READY, /* number is ready to use */
  FORM_CALC,  /* calculating */
  FORM_DIRTY  /* need calc */
};

enum {
  ERR_NONE,
  ERR_PARSE,
  ERR_RECURSE,
  ERR_DEPEND
};

/* each cell can contain:
  - no value, an empty cell
  - a text value
  - a numeric value
  - a sum of values (a formula starts with '=')
*/
typedef struct _CellData
{
  int parseError;
  double number;
  int isNumber;
  int calcFormula;
  char value[80];
  Operation operation;
  Cell dependencies[30]; /* cells that are depend on this cell */
  int dependenciesCount;
} CellData;


#define NUM_COL 26    /* A-Z=26 */
```

```
#define NUM_LIN 100  /* 0-99=100 */

static CellData data[NUM_LIN][NUM_COL];

static void cellDataInit(void)
{
  int i, j;
  for (i = 0; i < NUM_COL; i++)
  {
    for (j = 0; j < NUM_LIN; j++)
    {
      memset(&(data[i][j]), 0, sizeof(CellData));
    }
  }

  /* memset(data, 0, sizeof(data));  will also work because it is a constant array
}

static char* cellDataGetValue(int lin, int col)
{
  return data[lin][col].value;
}

static int cellDataIsFormula(int lin, int col)
{
  if (data[lin][col].isNumber || data[lin][col].value[0] != '=')
    return 0;
  else
    return 1;
}

static void cellDataSetDependenciesDirty(CellData* c)
{
  int i;
  for (i = 0; i < c->dependenciesCount; i++)
  {
    int dep_lin = c->dependencies[i].lin;
    int dep_col = c->dependencies[i].col;
    data[dep_lin][dep_col].calcFormula = FORM_DIRTY;
  }
}
```

```
static double cellDataIsNumber(int lin, int col)
{
  return data[lin][col].isNumber || (data[lin][col].value[0] == '=' && data[lin][co
}

static double cellDataGetNumber(int lin, int col);

static double cellDataSumCells(CellData* c)
{
  int i, j;
  double total = 0;

  for (i = c->operation.startLin; i <= c->operation.endLin; i++)
  {
    for (j = c->operation.startCol; j <= c->operation.endCol; j++)
    {
      if (cellDataIsNumber(i, j))
        total += cellDataGetNumber(i, j);
      else if (data[i][j].value[0] != 0)
      {
        c->parseError = ERR_DEPEND;
        return 0;
      }
    }
  }

  return total;
}

static void cellDataCalcFormula(CellData* c)
{
  if (c->calcFormula == FORM_CALC)
  {
    c->parseError = ERR_RECURSE;
    return;
  }

  cellDataSetDependenciesDirty(c);

  if (c->parseError == ERR_NONE)
  {
    switch (c->operation.type)
```

```
    {
    case OP_SUM:
      c->calcFormula = FORM_CALC;

      c->number = cellDataSumCells(c);

      if (c->parseError == ERR_NONE)
        c->calcFormula = FORM_READY;
      else
        c->calcFormula = FORM_DIRTY;
      break;
    }
  }
}

static double cellDataGetNumber(int lin, int col)
{
  CellData* c = &(data[lin][col]);
  if (c->value[0] == '=' && c->calcFormula != FORM_READY)
    cellDataCalcFormula(c);

  return c->number;
}

static char* cellDataGetString(int lin, int col)
{
  /* called when not a number */
  if (data[lin][col].value[0] == '=')
  {
    int error = data[lin][col].parseError;
    if (error != ERR_NONE)
    {
      if (error == ERR_PARSE)
        return "Error Invalid Formula!";
      else if (error == ERR_RECURSE)
        return "Error Recursive Formula!";
      else /* ERR_DEPEND */
        return "Error Dependency in Formula!";
    }
  }

  return data[lin][col].value;
```

```
}

static void cellDataSetNumber(int lin, int col, double value)
{
  CellData* c = &(data[lin][col]);

  c->isNumber = 1;
  c->number = value;
  c->calcFormula = FORM_READY;
  c->parseError = ERR_NONE;
  c->value[0] = 0;

  cellDataSetDependenciesDirty(c);
}

static void cellDataParseFormula(CellData* c)
{
  int l1, l2, n;
  char c1, c2, func[80];
  char *formula = c->value;

  c->parseError = ERR_PARSE;

  n = sscanf(formula+1, "%3s(%c%d:%c%d)", func, &c1, &l1, &c2, &l2);
  if (n != 5)
    return;

  if (strcmp(func, "sum") == 0)
    c->operation.type = OP_SUM;
  else
    return;

  if (c1 < 'A' || c1 > 'Z')
    return;
  if (c2 < 'A' || c2 > 'Z')
    return;
  if (l1 < 0 || l1 > 99)
    return;
  if (l2 < 0 || l2 > 99)
    return;

  c->parseError = ERR_NONE;
```

```
  c->operation.startCol = c1 - 'A';
  c->operation.startLin = l1;
  c->operation.endCol = c2 - 'A';
  c->operation.endLin = l2;
}

static void cellDataAddDependencies(CellData* c, int dep_lin, int dep_col)
{
  int i;
  Cell cell;
  for (i = 0; i < c->dependenciesCount; i++)
  {
    if (dep_lin == c->dependencies[i].lin &&
        dep_col == c->dependencies[i].col)
      return; /* already in the list */
  }

  /* not found, add to the list */
  cell.col = dep_col;
  cell.lin = dep_lin;
  c->dependencies[c->dependenciesCount] = cell;
  c->dependenciesCount++;
}

static void cellDataRemoveDependencies(CellData* c, int dep_lin, int dep_col)
{
  int i;
  for (i = 0; i < c->dependenciesCount; i++)
  {
    if (dep_lin == c->dependencies[i].lin &&
        dep_col == c->dependencies[i].col)
        break; /* found in the list */
  }

  if (i == c->dependenciesCount)
    return;

  /* remove from the list */
  for (; i < c->dependenciesCount - 1; i++)
  {
    c->dependencies[i] = c->dependencies[i+1];
```

```
  }

  c->dependenciesCount--;
}

static void cellDataUpdateDependencies(CellData* c, int lin, int col)
{
  int i, j;

  for (i = 0; i < NUM_LIN; i++)
  {
    for (j = 0; j < NUM_COL; j++)
    {
      if (i >= c->operation.startLin && i <= c->operation.endLin &&
          j >= c->operation.startCol && j <= c->operation.endCol)
        cellDataAddDependencies(&(data[i][j]), lin, col);
      else
        cellDataRemoveDependencies(&(data[i][j]), lin, col);
    }
  }
}

static void cellDataSetString(int lin, int col, const char *value)
{
  CellData* c = &(data[lin][col]);

  cellDataSetDependenciesDirty(c);

  c->isNumber = 0;
  c->number = 0;
  c->calcFormula = FORM_READY;
  c->parseError = ERR_NONE;
  strcpy(c->value, value);

  if (c->value[0] == '=')
  {
    c->calcFormula = FORM_DIRTY;

    cellDataParseFormula(c);

    if (c->parseError == ERR_NONE)
      cellDataUpdateDependencies(c, lin, col);
```

```
  }
}


/************************************ Interface ***********************************


static char* matrix_value_cb(Ihandle *self, int lin, int col)
{
  static char text[80];
  char *editcell = IupGetAttribute(self, "EDITCELL");
  if (editcell != NULL)
  {
    int elin, ecol;
    sscanf(editcell, "%d:%d", &elin, &ecol);
    if (elin == lin && ecol == col && cellDataIsFormula(lin - 1, col - 1))
      return cellDataGetValue(lin - 1, col - 1);
  }

  if (lin == 0 && col == 0)
    return "";
  else if (lin == 0)
  {
    sprintf(text, "%c", 'A' + col - 1);
    return text;
  }
  else if (col == 0)
  {
    sprintf(text, "%d", lin - 1);
    return text;
  }

  if (cellDataIsNumber(lin - 1, col - 1))
  {
    double number = cellDataGetNumber(lin - 1, col - 1);
    if (cellDataIsNumber(lin - 1, col - 1))
      sprintf(text, "%.2lf", number);
    else
      return cellDataGetString(lin - 1, col - 1);
  }
  else
    return cellDataGetString(lin - 1, col - 1);
```

```
  return text;
}

static int matrix_value_edit_cb(Ihandle *self, int lin, int col, char* newvalue)
{
  char str[80];
  double d;

  int i = sscanf(newvalue, "%lf%s", &d, &str);
  if (i == 1) /* check if there is more text after the number (text may start with
    cellDataSetNumber(lin - 1, col - 1, d);
  else
    cellDataSetString(lin - 1, col - 1, newvalue);

  (void)self;
  return IUP_DEFAULT;
}

int main(int argc, char **argv)
{
  Ihandle *dlg, *matrix;

  IupOpen(&argc, &argv);
  IupControlsOpen();

  matrix = IupMatrix(NULL);

  IupSetAttribute(matrix, "NAME", "MATRIX");

  IupSetInt(matrix, "NUMCOL", NUM_COL);
  IupSetInt(matrix, "NUMLIN", NUM_LIN);

  IupSetAttribute(matrix, "NUMCOL_VISIBLE", "4");
  IupSetAttribute(matrix, "NUMLIN_VISIBLE", "7");

  IupSetAttribute(matrix, "WIDTH0", "15");
  IupSetAttribute(matrix, "WIDTHDEF", "40");
  IupSetAttribute(matrix, "HEIGHT0", "8");
  IupSetAttribute(matrix, "RESIZEMATRIX", "Yes");

  cellDataInit();
```

```
  IupSetCallback(matrix, "VALUE_CB", (Icallback)matrix_value_cb);
  IupSetCallback(matrix, "VALUE_EDIT_CB", (Icallback)matrix_value_edit_cb);

  dlg = IupDialog(matrix);
  IupSetAttribute(dlg, "TITLE", "Cells");
  IupSetAttribute(dlg, "SIZE", "300x150");

  IupShowXY(dlg, IUP_CENTER, IUP_CENTER);

  IupMainLoop();

  IupClose();

  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")

NUM_COL = 26    -- A-Z=26
NUM_LIN = 100   -- 0-99=100

FORM_READY = 0  -- number is ready to use
FORM_CALC = 1   -- calculating
FORM_DIRTY = 2  -- need calc

ERR_NONE = 0
ERR_PARSE = 1
ERR_RECURSE = 2
ERR_DEPEND = 3

data = {}

function cellDataInit()
  for i = 1, NUM_LIN do
    data[i] = {}
    for j = 1, NUM_COL do
      data[i][j] = { parseError = 0, number = 0.0, isNumber = nil, calcFormula = 0,
                operation = { type = 0, startLin = 0, startCol = 0, endLin = 0, endC
                dependencies = {}, dependenciesCount = 0 }
    end
```

```
  end
end

function cellDataGetValue(lin, col)
  return data[lin][col].value
end

function cellDataIsFormula(lin, col)
  if data[lin][col].isNumber or (data[lin][col].value and not string.match(data[lin
    return nil
  else
    return 1
  end
end

function cellDataSetDependenciesDirty(lin, col)
  local cellData = data[lin][col]
  for i = 1, cellData.dependenciesCount do
    local dep_lin = cellData.dependencies[i].lin
    local dep_col = cellData.dependencies[i].col
    data[dep_lin][dep_col].calcFormula = FORM_DIRTY
  end
end

function cellDataIsNumber(lin, col)
  return data[lin][col].isNumber or ((data[lin][col].value and string.match(data[li
end

function cellDataCalcFormula(lin, col)
  if data[lin][col].calcFormula == FORM_CALC then
    data[lin][col].parseError = ERR_RECURSE
    return
  end

  cellDataSetDependenciesDirty(lin, col)

  if data[lin][col].parseError == ERR_NONE then
    if data[lin][col].operation.type == OP_SUM then
      data[lin][col].calcFormula = FORM_CALC

      data[lin][col].number = cellDataSumCells(lin, col)
```

```
      if data[lin][col].parseError == ERR_NONE then
        data[lin][col].calcFormula = FORM_READY
      else
        data[lin][col].calcFormula = FORM_DIRTY
      end
    end
  end
end

function cellDataSumCells(lin, col)
  local total = 0
  for i = data[lin][col].operation.startLin, data[lin][col].operation.endLin do
    for j = data[lin][col].operation.startCol, data[lin][col].operation.endCol do
      if cellDataIsNumber(i, j) then
        total = total + cellDataGetNumber(i, j)
      elseif data[i][j].value then
        data[lin][col].parseError = ERR_DEPEND
        return 0
      end
    end
  end
  return total
end

function cellDataGetNumber(lin, col)
  if data[lin][col].value and string.match(data[lin][col].value, "%s*=%s*") and dat
    cellDataCalcFormula(lin, col)
  end

  return data[lin][col].number
end

function cellDataGetString(lin, col)
  -- called when not a number
  if data[lin][col].value and string.match(data[lin][col].value, "%s*=%s*") then
    local error = data[lin][col].parseError
    if error ~= ERR_NONE then
      if error == ERR_PARSE then
        return "Error␣Invalid␣Formula!"
      elseif error == ERR_RECURSE then
        return "Error␣Recursive␣Formula!"
      else -- ERR_DEPEND
```

```
        return "Error Dependency in Formula!"
      end
    end
  end

  return data[lin][col].value
end

function cellDataSetNumber(lin, col, value)
  data[lin][col].isNumber = 1
  data[lin][col].number = value
  data[lin][col].calcFormula = FORM_READY
  data[lin][col].parseError = ERR_NONE
  data[lin][col].value = nil
  cellDataSetDependenciesDirty(lin, col)
end

function cellDataParseFormula(lin, col)
  local formula = data[lin][col].value

  data[lin][col].parseError = ERR_PARSE

  local op, c1, l1, c2, l2 = string.match(formula, "%s*=%s*(%l+)%((%u)(%d):(%u)(%d)%
  if not op or not c1 or not l1 or not c2 or not l2 then
    return
  end

  if op == "sum" then
    data[lin][col].operation.type = OP_SUM
  else
    return
  end

  if c1 < "A" or c1 > "Z" then
    return
  end
  if c2 < "A" or c2 > "Z" then
    return
  end
  if tonumber(l1) < 0 or tonumber(l1) > 99 then
    return
  end
```

```
  if tonumber(l2) < 0 or tonumber(l2) > 99 then
    return
  end

  data[lin][col].parseError = ERR_NONE

  data[lin][col].operation.startCol = string.byte(c1) - string.byte('A') + 1
  data[lin][col].operation.startLin = tonumber(l1)
  data[lin][col].operation.endCol = string.byte(c2) - string.byte('A') + 1
  data[lin][col].operation.endLin = tonumber(l2)
end

function cellDataAddDependencies(lin, col, dep_lin, dep_col)
  for i = 1, data[lin][col].dependenciesCount do
    if dep_lin == data[lin][col].dependencies[i].lin and
      dep_col == data[lin][col].dependencies[i].col then
      return -- already in the list
    end
  end

  -- not found, add to the list
  data[lin][col].dependenciesCount = data[lin][col].dependenciesCount + 1
  data[lin][col].dependencies[data[lin][col].dependenciesCount] = {}
  data[lin][col].dependencies[data[lin][col].dependenciesCount].lin = dep_lin
  data[lin][col].dependencies[data[lin][col].dependenciesCount].col = dep_col
end

function cellDataRemoveDependencies(lin, col, dep_lin, dep_col)
  local index = 0
  for i = 1, data[lin][col].dependenciesCount do
    if dep_lin == data[lin][col].dependencies[i].lin and
      dep_col == data[lin][col].dependencies[i].col then
      index = i
      break -- found in the list
    end
  end

  if index == data[lin][col].dependenciesCount then
    return
  end

  -- remove from the list
```

```
  for i = index, data[lin][col].dependenciesCount - 1 do
    data[lin][col].dependencies[i] = data[lin][col].dependencies[i+1]
  end

  data[lin][col].dependenciesCount = data[lin][col].dependenciesCount - 1
end

function cellDataUpdateDependencies(lin, col)
  for i = 1, NUM_LIN do
    for j = 1, NUM_COL do
      if i >= data[lin][col].operation.startLin and i <= data[lin][col].operation.e
        j >= data[lin][col].operation.startCol and j <= data[lin][col].operation.en
        cellDataAddDependencies(i, j, lin, col)
      else
        cellDataRemoveDependencies(i, j, lin, col)
      end
    end
  end
end

function cellDataSetString(lin, col, value)
  cellDataSetDependenciesDirty(lin, col)

  data[lin][col].isNumber = nil
  data[lin][col].number = 0
  data[lin][col].calcFormula = FORM_READY
  data[lin][col].parseError = ERR_NONE
  data[lin][col].value = value

  if string.match(data[lin][col].value, "%s*=%s*") then
    data[lin][col].calcFormula = FORM_DIRTY

    cellDataParseFormula(lin, col)

    if data[lin][col].parseError == ERR_NONE then
      cellDataUpdateDependencies(lin, col)
    end
  end
end

--************************************ Interface ********************************
```

```
matrix = iup.matrix{numcol = NUM_COL, numlin = NUM_LIN, numcol_visible = 4, numlin_
          width0 = 15, height0 = 8, resizematrix = "YES"}

dlg = iup.dialog{iup.hbox{matrix}; title = "Cells", size = "300x150"}

cellDataInit()

function matrix:value_cb(lin, col)
  local text
  local editcell = self.editcell
  if editcell then
    local elin, ecol = string.match(editcell, "(%d):(%d)")
    if tonumber(elin) == lin and tonumber(ecol) == col and cellDataIsFormula(lin, c
      return cellDataGetValue(lin, col)
    end
  end

  if lin == 0 and col == 0 then
    return ""
  elseif lin == 0 then
    return string.format("%c", string.byte("A") + col - 1)
  elseif col == 0 then
    return string.format("%d", lin)
  end

  if cellDataIsNumber(lin, col) then
    local numberValue = cellDataGetNumber(lin, col)
    if not numberValue or not cellDataIsNumber(lin, col) then
      text = cellDataGetString(lin, col)
    else
      text = string.format("%.2f", numberValue)
    end
  else
    text = cellDataGetString(lin, col)
  end

  return text
end

function matrix:value_edit_cb(lin, col, newValue)
  if string.find(newValue, "^[+-]?%d+$") then -- check if there is more text after
    cellDataSetNumber(lin, col, newValue)
```

```
  else
    cellDataSetString(lin, col, newValue)
  end
end

dlg:showxy( iup.CENTER, iup.CENTER )

if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
end
```

单元格 *B0* 的值为 *=sum(B1:C4)*。单元格的值可以为空，可以是一个数字，一个字符串，或是一个公式。公式以 = 开始。但本例的程序只支持 *=sum(XY:XY)* 这个公式。

C语言

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

#include <iup.h>
#include <iupcontrols.h>

enum { OP_SUM };

typedef struct _Cell
{
  int lin;
  int col;
} Cell;

typedef struct _Operation
{
  int type;
  int startLin;
  int endLin;
  int startCol;
  int endCol;
} Operation;

enum
{
  FORM_READY, /* number is ready to use */
  FORM_CALC,  /* calculating */
  FORM_DIRTY  /* need calc */
```

```
};

enum {
  ERR_NONE,
  ERR_PARSE,
  ERR_RECURSE,
  ERR_DEPEND
};

/* each cell can contain:
  - no value, an empty cell
  - a text value
  - a numeric value
  - a sum of values (a formula starts with '=')
*/
typedef struct _CellData
{
  int parseError;
  double number;
  int isNumber;
  int calcFormula;
  char value[80];
  Operation operation;
  Cell dependencies[30]; /* cells that are depend on this cell */
  int dependenciesCount;
} CellData;


#define NUM_COL 26    /* A-Z=26 */
#define NUM_LIN 100   /* 0-99=100 */

static CellData data[NUM_LIN][NUM_COL];

static void cellDataInit(void)
{
  int i, j;
  for (i = 0; i < NUM_COL; i++)
  {
    for (j = 0; j < NUM_LIN; j++)
    {
      memset(&(data[i][j]), 0, sizeof(CellData));
    }
```

```
  }

  /* memset(data, 0, sizeof(data));  will also work because it is a constant array
}

static char* cellDataGetValue(int lin, int col)
{
  return data[lin][col].value;
}

static int cellDataIsFormula(int lin, int col)
{
  if (data[lin][col].isNumber || data[lin][col].value[0] != '=')
    return 0;
  else
    return 1;
}

static void cellDataSetDependenciesDirty(CellData* c)
{
  int i;
  for (i = 0; i < c->dependenciesCount; i++)
  {
    int dep_lin = c->dependencies[i].lin;
    int dep_col = c->dependencies[i].col;
    data[dep_lin][dep_col].calcFormula = FORM_DIRTY;
  }
}

static double cellDataIsNumber(int lin, int col)
{
  return data[lin][col].isNumber || (data[lin][col].value[0] == '=' && data[lin][co
}

static double cellDataGetNumber(int lin, int col);

static double cellDataSumCells(CellData* c)
{
  int i, j;
  double total = 0;

  for (i = c->operation.startLin; i <= c->operation.endLin; i++)
```

```
  {
    for (j = c->operation.startCol; j <= c->operation.endCol; j++)
    {
      if (cellDataIsNumber(i, j))
        total += cellDataGetNumber(i, j);
      else if (data[i][j].value[0] != 0)
      {
        c->parseError = ERR_DEPEND;
        return 0;
      }
    }
  }

  return total;
}

static void cellDataCalcFormula(CellData* c)
{
  if (c->calcFormula == FORM_CALC)
  {
    c->parseError = ERR_RECURSE;
    return;
  }

  cellDataSetDependenciesDirty(c);

  if (c->parseError == ERR_NONE)
  {
    switch (c->operation.type)
    {
    case OP_SUM:
      c->calcFormula = FORM_CALC;

      c->number = cellDataSumCells(c);

      if (c->parseError == ERR_NONE)
        c->calcFormula = FORM_READY;
      else
        c->calcFormula = FORM_DIRTY;
      break;
    }
  }
```

```
}

static double cellDataGetNumber(int lin, int col)
{
  CellData* c = &(data[lin][col]);
  if (c->value[0] == '=' && c->calcFormula != FORM_READY)
    cellDataCalcFormula(c);

  return c->number;
}

static char* cellDataGetString(int lin, int col)
{
  /* called when not a number */
  if (data[lin][col].value[0] == '=')
  {
    int error = data[lin][col].parseError;
    if (error != ERR_NONE)
    {
      if (error == ERR_PARSE)
        return "Error Invalid Formula!";
      else if (error == ERR_RECURSE)
        return "Error Recursive Formula!";
      else /* ERR_DEPEND */
        return "Error Dependency in Formula!";
    }
  }

  return data[lin][col].value;
}

static void cellDataSetNumber(int lin, int col, double value)
{
  CellData* c = &(data[lin][col]);

  c->isNumber = 1;
  c->number = value;
  c->calcFormula = FORM_READY;
  c->parseError = ERR_NONE;
  c->value[0] = 0;

  cellDataSetDependenciesDirty(c);
```

```
}

static void cellDataParseFormula(CellData* c)
{
  int l1, l2, n;
  char c1, c2, func[80];
  char *formula = c->value;

  c->parseError = ERR_PARSE;

  n = sscanf(formula+1, "%3s(%c%d:%c%d)", func, &c1, &l1, &c2, &l2);
  if (n != 5)
    return;

  if (strcmp(func, "sum") == 0)
    c->operation.type = OP_SUM;
  else
    return;

  if (c1 < 'A' || c1 > 'Z')
    return;
  if (c2 < 'A' || c2 > 'Z')
    return;
  if (l1 < 0 || l1 > 99)
    return;
  if (l2 < 0 || l2 > 99)
    return;

  c->parseError = ERR_NONE;

  c->operation.startCol = c1 - 'A';
  c->operation.startLin = l1;
  c->operation.endCol = c2 - 'A';
  c->operation.endLin = l2;
}

static void cellDataAddDependencies(CellData* c, int dep_lin, int dep_col)
{
  int i;
  Cell cell;
  for (i = 0; i < c->dependenciesCount; i++)
  {
```

```
    if (dep_lin == c->dependencies[i].lin &&
        dep_col == c->dependencies[i].col)
      return; /* already in the list */
  }

  /* not found, add to the list */
  cell.col = dep_col;
  cell.lin = dep_lin;
  c->dependencies[c->dependenciesCount] = cell;
  c->dependenciesCount++;
}

static void cellDataRemoveDependencies(CellData* c, int dep_lin, int dep_col)
{
  int i;
  for (i = 0; i < c->dependenciesCount; i++)
  {
    if (dep_lin == c->dependencies[i].lin &&
        dep_col == c->dependencies[i].col)
        break; /* found in the list */
  }

  if (i == c->dependenciesCount)
    return;

  /* remove from the list */
  for (; i < c->dependenciesCount - 1; i++)
  {
    c->dependencies[i] = c->dependencies[i+1];
  }

  c->dependenciesCount--;
}

static void cellDataUpdateDependencies(CellData* c, int lin, int col)
{
  int i, j;

  for (i = 0; i < NUM_LIN; i++)
  {
    for (j = 0; j < NUM_COL; j++)
    {
```

```
      if (i >= c->operation.startLin && i <= c->operation.endLin &&
          j >= c->operation.startCol && j <= c->operation.endCol)
        cellDataAddDependencies(&(data[i][j]), lin, col);
      else
        cellDataRemoveDependencies(&(data[i][j]), lin, col);
    }
  }
}

static void cellDataSetString(int lin, int col, const char *value)
{
  CellData* c = &(data[lin][col]);

  cellDataSetDependenciesDirty(c);

  c->isNumber = 0;
  c->number = 0;
  c->calcFormula = FORM_READY;
  c->parseError = ERR_NONE;
  strcpy(c->value, value);

  if (c->value[0] == '=')
  {
    c->calcFormula = FORM_DIRTY;

    cellDataParseFormula(c);

    if (c->parseError == ERR_NONE)
      cellDataUpdateDependencies(c, lin, col);
  }
}


/************************************ Interface ********************************


static char* matrix_value_cb(Ihandle *self, int lin, int col)
{
  static char text[80];
  char *editcell = IupGetAttribute(self, "EDITCELL");
  if (editcell != NULL)
  {
```

```
    int elin, ecol;
    sscanf(editcell, "%d:%d", &elin, &ecol);
    if (elin == lin && ecol == col && cellDataIsFormula(lin - 1, col - 1))
      return cellDataGetValue(lin - 1, col - 1);
  }

  if (lin == 0 && col == 0)
    return "";
  else if (lin == 0)
  {
    sprintf(text, "%c", 'A' + col - 1);
    return text;
  }
  else if (col == 0)
  {
    sprintf(text, "%d", lin - 1);
    return text;
  }

  if (cellDataIsNumber(lin - 1, col - 1))
  {
    double number = cellDataGetNumber(lin - 1, col - 1);
    if (cellDataIsNumber(lin - 1, col - 1))
      sprintf(text, "%.2lf", number);
    else
      return cellDataGetString(lin - 1, col - 1);
  }
  else
    return cellDataGetString(lin - 1, col - 1);

  return text;
}

static int matrix_value_edit_cb(Ihandle *self, int lin, int col, char* newvalue)
{
  char str[80];
  double d;

  int i = sscanf(newvalue, "%lf%s", &d, &str);
  if (i == 1) /* check if there is more text after the number (text may start with
    cellDataSetNumber(lin - 1, col - 1, d);
  else
```

```
    cellDataSetString(lin - 1, col - 1, newvalue);

  (void)self;
  return IUP_DEFAULT;
}

int main(int argc, char **argv)
{
  Ihandle *dlg, *matrix;

  IupOpen(&argc, &argv);
  IupControlsOpen();

  matrix = IupMatrix(NULL);

  IupSetAttribute(matrix, "NAME", "MATRIX");

  IupSetInt(matrix, "NUMCOL", NUM_COL);
  IupSetInt(matrix, "NUMLIN", NUM_LIN);

  IupSetAttribute(matrix, "NUMCOL_VISIBLE", "4");
  IupSetAttribute(matrix, "NUMLIN_VISIBLE", "7");

  IupSetAttribute(matrix, "WIDTH0", "15");
  IupSetAttribute(matrix, "WIDTHDEF", "40");
  IupSetAttribute(matrix, "HEIGHT0", "8");
  IupSetAttribute(matrix, "RESIZEMATRIX", "Yes");

  cellDataInit();

  IupSetCallback(matrix, "VALUE_CB", (Icallback)matrix_value_cb);
  IupSetCallback(matrix, "VALUE_EDIT_CB", (Icallback)matrix_value_edit_cb);

  dlg = IupDialog(matrix);
  IupSetAttribute(dlg, "TITLE", "Cells");
  IupSetAttribute(dlg, "SIZE", "300x150");

  IupShowXY(dlg, IUP_CENTER, IUP_CENTER);

  IupMainLoop();

  IupClose();
```

```
  return EXIT_SUCCESS;
}
```

Lua语言

```
require("iuplua")

NUM_COL = 26    -- A-Z=26
NUM_LIN = 100   -- 0-99=100

FORM_READY = 0  -- number is ready to use
FORM_CALC = 1   -- calculating
FORM_DIRTY = 2  -- need calc

ERR_NONE = 0
ERR_PARSE = 1
ERR_RECURSE = 2
ERR_DEPEND = 3

data = {}

function cellDataInit()
  for i = 1, NUM_LIN do
    data[i] = {}
    for j = 1, NUM_COL do
      data[i][j] = { parseError = 0, number = 0.0, isNumber = nil, calcFormula = 0,
                operation = { type = 0, startLin = 0, startCol = 0, endLin = 0, endC
                dependencies = {}, dependenciesCount = 0 }
    end
  end
end

function cellDataGetValue(lin, col)
  return data[lin][col].value
end

function cellDataIsFormula(lin, col)
  if data[lin][col].isNumber or (data[lin][col].value and not string.match(data[lin
    return nil
  else
    return 1
  end
```

```
end

function cellDataSetDependenciesDirty(lin, col)
  local cellData = data[lin][col]
  for i = 1, cellData.dependenciesCount do
    local dep_lin = cellData.dependencies[i].lin
    local dep_col = cellData.dependencies[i].col
    data[dep_lin][dep_col].calcFormula = FORM_DIRTY
  end
end

function cellDataIsNumber(lin, col)
  return data[lin][col].isNumber or ((data[lin][col].value and string.match(data[li
end

function cellDataCalcFormula(lin, col)
  if data[lin][col].calcFormula == FORM_CALC then
    data[lin][col].parseError = ERR_RECURSE
    return
  end

  cellDataSetDependenciesDirty(lin, col)

  if data[lin][col].parseError == ERR_NONE then
    if data[lin][col].operation.type == OP_SUM then
      data[lin][col].calcFormula = FORM_CALC

      data[lin][col].number = cellDataSumCells(lin, col)

      if data[lin][col].parseError == ERR_NONE then
        data[lin][col].calcFormula = FORM_READY
      else
        data[lin][col].calcFormula = FORM_DIRTY
      end
    end
  end
end

function cellDataSumCells(lin, col)
  local total = 0
  for i = data[lin][col].operation.startLin, data[lin][col].operation.endLin do
    for j = data[lin][col].operation.startCol, data[lin][col].operation.endCol do
```

```
      if cellDataIsNumber(i, j) then
        total = total + cellDataGetNumber(i, j)
      elseif data[i][j].value then
        data[lin][col].parseError = ERR_DEPEND
        return 0
      end
    end
  end
  return total
end

function cellDataGetNumber(lin, col)
  if data[lin][col].value and string.match(data[lin][col].value, "%s*=%s*") and dat
    cellDataCalcFormula(lin, col)
  end

  return data[lin][col].number
end

function cellDataGetString(lin, col)
  -- called when not a number
  if data[lin][col].value and string.match(data[lin][col].value, "%s*=%s*") then
    local error = data[lin][col].parseError
    if error ~= ERR_NONE then
      if error == ERR_PARSE then
        return "Error Invalid Formula!"
      elseif error == ERR_RECURSE then
        return "Error Recursive Formula!"
      else -- ERR_DEPEND
        return "Error Dependency in Formula!"
      end
    end
  end

  return data[lin][col].value
end

function cellDataSetNumber(lin, col, value)
  data[lin][col].isNumber = 1
  data[lin][col].number = value
  data[lin][col].calcFormula = FORM_READY
  data[lin][col].parseError = ERR_NONE
```

```
  data[lin][col].value = nil
  cellDataSetDependenciesDirty(lin, col)
end

function cellDataParseFormula(lin, col)
  local formula = data[lin][col].value

  data[lin][col].parseError = ERR_PARSE

  local op, c1, l1, c2, l2 = string.match(formula, "%s*=%s*(%l+)%((%u)(%d):(%u)(%d)%
  if not op or not c1 or not l1 or not c2 or not l2 then
    return
  end

  if op == "sum" then
    data[lin][col].operation.type = OP_SUM
  else
    return
  end

  if c1 < "A" or c1 > "Z" then
    return
  end
  if c2 < "A" or c2 > "Z" then
    return
  end
  if tonumber(l1) < 0 or tonumber(l1) > 99 then
    return
  end
  if tonumber(l2) < 0 or tonumber(l2) > 99 then
    return
  end

  data[lin][col].parseError = ERR_NONE

  data[lin][col].operation.startCol = string.byte(c1) - string.byte('A') + 1
  data[lin][col].operation.startLin = tonumber(l1)
  data[lin][col].operation.endCol = string.byte(c2) - string.byte('A') + 1
  data[lin][col].operation.endLin = tonumber(l2)
end

function cellDataAddDependencies(lin, col, dep_lin, dep_col)
```

```
  for i = 1, data[lin][col].dependenciesCount do
    if dep_lin == data[lin][col].dependencies[i].lin and
      dep_col == data[lin][col].dependencies[i].col then
      return -- already in the list
    end
  end

  -- not found, add to the list
  data[lin][col].dependenciesCount = data[lin][col].dependenciesCount + 1
  data[lin][col].dependencies[data[lin][col].dependenciesCount] = {}
  data[lin][col].dependencies[data[lin][col].dependenciesCount].lin = dep_lin
  data[lin][col].dependencies[data[lin][col].dependenciesCount].col = dep_col
end

function cellDataRemoveDependencies(lin, col, dep_lin, dep_col)
  local index = 0
  for i = 1, data[lin][col].dependenciesCount do
    if dep_lin == data[lin][col].dependencies[i].lin and
      dep_col == data[lin][col].dependencies[i].col then
      index = i
      break -- found in the list
    end
  end

  if index == data[lin][col].dependenciesCount then
    return
  end

  -- remove from the list
  for i = index, data[lin][col].dependenciesCount - 1 do
    data[lin][col].dependencies[i] = data[lin][col].dependencies[i+1]
  end

  data[lin][col].dependenciesCount = data[lin][col].dependenciesCount - 1
end

function cellDataUpdateDependencies(lin, col)
  for i = 1, NUM_LIN do
    for j = 1, NUM_COL do
      if i >= data[lin][col].operation.startLin and i <= data[lin][col].operation.e
        j >= data[lin][col].operation.startCol and j <= data[lin][col].operation.en
        cellDataAddDependencies(i, j, lin, col)
```

```
      else
        cellDataRemoveDependencies(i, j, lin, col)
      end
    end
  end
end

function cellDataSetString(lin, col, value)
  cellDataSetDependenciesDirty(lin, col)

  data[lin][col].isNumber = nil
  data[lin][col].number = 0
  data[lin][col].calcFormula = FORM_READY
  data[lin][col].parseError = ERR_NONE
  data[lin][col].value = value

  if string.match(data[lin][col].value, "%s*=%s*") then
    data[lin][col].calcFormula = FORM_DIRTY

    cellDataParseFormula(lin, col)

    if data[lin][col].parseError == ERR_NONE then
      cellDataUpdateDependencies(lin, col)
    end
  end
end

--************************************ Interface ********************************

matrix = iup.matrix{numcol = NUM_COL, numlin = NUM_LIN, numcol_visible = 4, numlin_
          width0 = 15, height0 = 8, resizematrix = "YES"}

dlg = iup.dialog{iup.hbox{matrix}; title = "Cells", size = "300x150"}

cellDataInit()

function matrix:value_cb(lin, col)
  local text
  local editcell = self.editcell
  if editcell then
    local elin, ecol = string.match(editcell, "(%d):(%d)")
    if tonumber(elin) == lin and tonumber(ecol) == col and cellDataIsFormula(lin, c
```

```
      return cellDataGetValue(lin, col)
    end
  end

  if lin == 0 and col == 0 then
    return ""
  elseif lin == 0 then
    return string.format("%c", string.byte("A") + col - 1)
  elseif col == 0 then
    return string.format("%d", lin)
  end

  if cellDataIsNumber(lin, col) then
    local numberValue = cellDataGetNumber(lin, col)
    if not numberValue or not cellDataIsNumber(lin, col) then
      text = cellDataGetString(lin, col)
    else
      text = string.format("%.2f", numberValue)
    end
  else
    text = cellDataGetString(lin, col)
  end

  return text
end

function matrix:value_edit_cb(lin, col, newValue)
  if string.find(newValue, "^[+-]?%d+$") then -- check if there is more text after
    cellDataSetNumber(lin, col, newValue)
  else
    cellDataSetString(lin, col, newValue)
  end
end

dlg:showxy( iup.CENTER, iup.CENTER )

if (iup.MainLoopLevel()==0) then
  iup.MainLoop()
end
```

我们使用 *IupMatrix* 控件来实现一个非常简单的表格程序。它只支持累加单元格的值。目前 *IupMatrixEx* 元素已经支持了公式。