

Sumário

GLADE - Um breve tutorial.....	3
Créditos.....	3
Apresentando o Glade como uma IDE.....	3
A área de trabalho do Glade.....	3
1 - Janela Principal de Gerenciamento de Projeto.....	4
2 - Paleta de componentes.....	4
3 - Caixa de propriedades.....	5
4 - Árvore de Widgets.....	6
5 - Área de Transferência.....	7
Objetos do GTK.....	7
Libglade.....	11
Funções básicas.....	11
Função gtk_init().....	12
Função gtk_main().....	12
Função Glade_init().....	12
Função Glade_xml_new().....	12
Função Glade_xml_signal_autoconnect().....	13
Função Glade_xml_get_widget().....	13
Hello World em libglade.....	13
Começando.....	14
Criando a interface com Glade.....	15
Criando a janela.....	15
Dividindo a Janela pra incluir os objetos.....	15
Os outros objetos.....	16
Perfumaria!.....	17
Janela.....	17
Rótulo.....	17
Botões.....	18
Salvando a interface pela primeira vez.....	18
O código fonte básico e comentado.....	19
Compilando.....	20
Dando vida ao programa.....	22
Protótipo dos Manipuladores Eventos.....	23
As funções manipuladoras.....	23
Conclusão.....	25
APENDICE I - Simplificando as listas encadeadas com a GList.....	26
Listas duplamente encadeadas em C.....	27
O tipo GList.....	27
Funções Típicas para manipular a glist.....	28
Funções de inclusão.....	28
Funções de remoção.....	29
Outras funções.....	29
A GList na prática.....	29
Inclusão de elementos na lista.....	31
Quanto itens há na lista?.....	31
Emitindo um aviso em caso de falha.....	31
Exibindo o que há na lista.....	32
Ordenando a lista.....	33
Invertendo a lista.....	35
Removendo um elemento da lista.....	35

Conclusão.....	38
Encerramento.....	38
Marcas registradas.....	38
Licença de Documentação Livre GNU.....	39
GNU Free Documentation License.....	39

GLADE - Um breve tutorial

O Objetivo desta obra é orientar ao novos programadores que usarão Glade/GTK como suas ferramentas de trabalho.

Programar em Glade/GTK é muito simples e em qualquer máquina com poucos recursos é possível desenvolver com eles.

Os requisitos mínimos são a posse de um computador com Linux, bibliotecas GTK e e suas dependências o ambiente de desenvolvimento Glade e algum editor de textos, preferencialmente que seja apropriado pra programação que permitem o destaque de comandos, eu sugiro o Diasce 2, ou o Anjuta, mas pra fins de simplificação esta obra usará o bom e velho gedit que está na maioria das distros GNU/Linux com o Gnome instalado.

Maiores informações no capítulo de Encerramento.

Boa Leitura!

Créditos

Copyright (c) 2004 Wellington Rodrigues Braga (www.gtk-br.cjb.net).
É dada permissão para copiar, distribuir e/ou modificar este documento sob os termos da Licença de Documentação Livre GNU, Versão 1.1 ou qualquer versão posterior publicada pela Free Software Foundation; sem Seções Invariantes, sem Capa da Frente, e sem Textos da Quarta-Capa .
Uma cópia da licença em está inclusa na seção intitulada
'Licença de Documentação Livre GNU'.

Apresentando o Glade como uma IDE

Glade (<http://glade.gnome.org>) é uma ferramenta livre para construir interfaces para o GTK+ (<http://www.gtk.org>) e Gnome (<http://www.gnome.org>). Distribuído sob a licença GNU/GPL, poderá ser livremente redistribuído e modificado.

O Glade pode produzir código fonte em C. Mas C++, Ada95, Python e Perl também está disponível, através de ferramentas externas que manipulam o arquivo XML gerado por ele.

O Site Oficial pode ser encontrado no endereço: <http://glade.gnome.org> e a última versão para download que geralmente é lançada logo após atualizações nas bibliotecas do GTK+.

Ele pode ser obtido neste mesmo endereço, no repositório oficial ou no CD da sua distribuição Linux preferida.

A área de trabalho do Glade

O Glade não possui uma área de trabalho como no Visual Basic. Ele é mais parecido com o Borland C++ (nem tanto, é claro :-)) e é dividido em 5 janelas:

1. Janela de Gerenciamento de Projetos
2. Paleta de Componentes

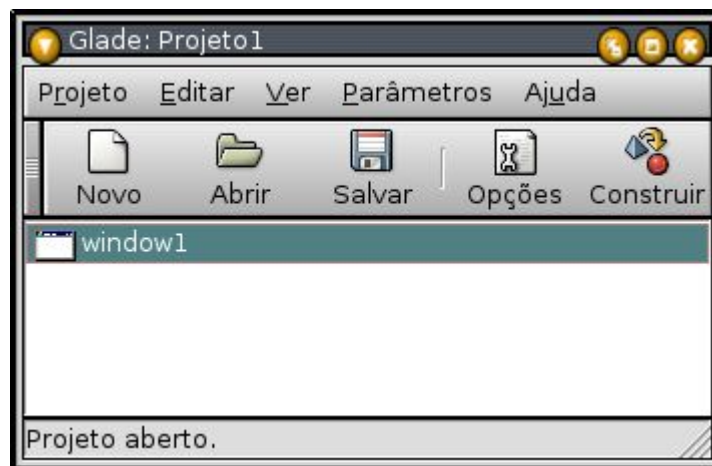
3. Caixa de Propriedades
4. Árvore de Widgets
5. Área de Transferência

1 - Janela Principal de Gerenciamento de Projeto

Como o nome diz esta é a janela principal do Glade, pois é a partir daqui onde poderemos Criar novo, Abrir, Salvar e Configurar o Projeto além de gerar o código fonte etc.

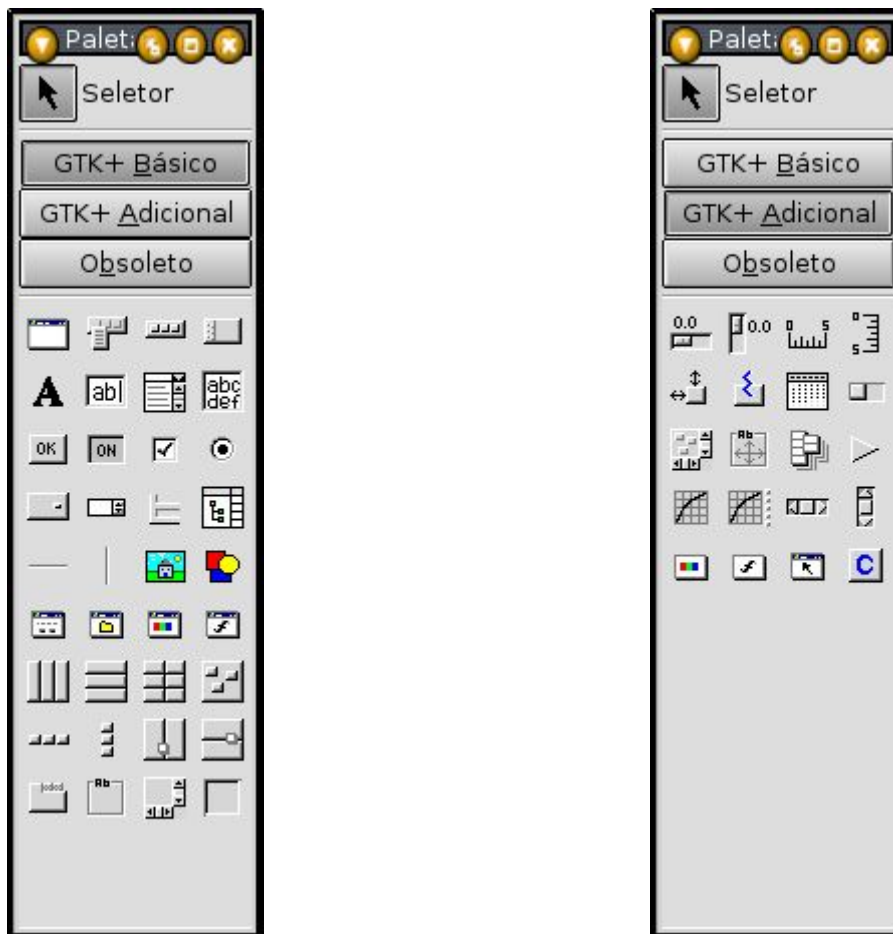
Este texto parte da idéia que as operações básicas (Novo, Abrir, Salvar, Copiar, Recortar, Colar) já são conhecidas do leitor, uma vez que isto é igual a qualquer programa convencional com estas funções.

É importante ressaltar que as outras janelas poderão ser exibidas a partir do menu VER desta janela.



2 - Paleta de componentes

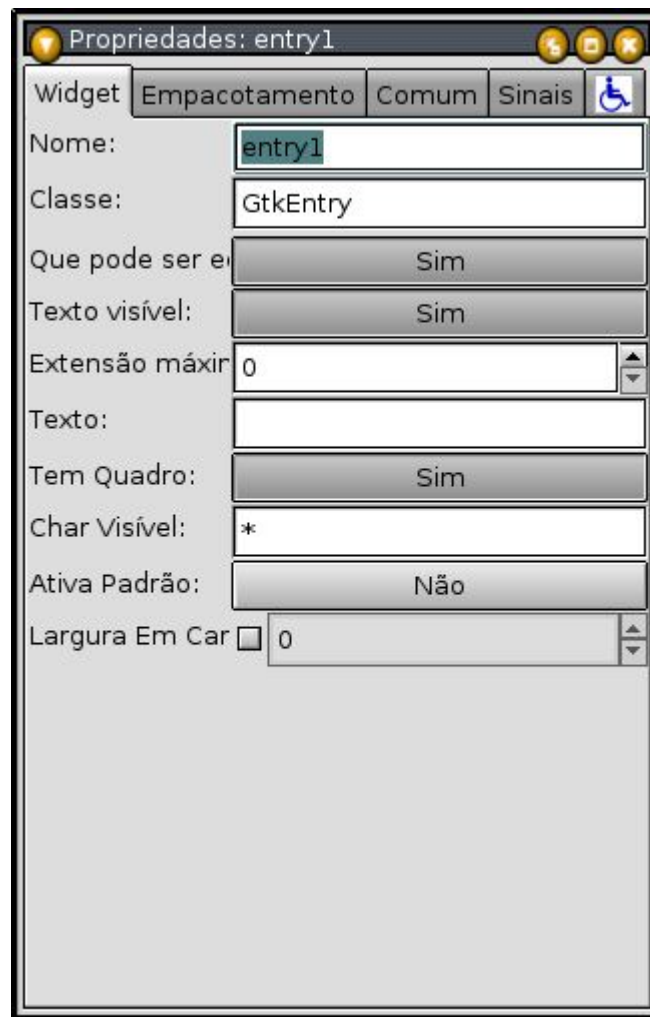
A paleta de componentes é onde estão todos (ou quase todos) os componentes que poderão ser usados no desenvolvimento visual da aplicação. Desde a janela principal até os “poderosos” rótulos e botões, passando pelas caixas combinação e menus, entre outros.



Observe que eu omiti a imagem da paleta “Obsoletos”, pois como o próprio nome já deixa claro há coisas mais úteis do que aqueles objetos. E a paleta Gnome, também foi omitida por questão de comodismo, apenas; uma vez que ela só aparecerá se você iniciar um projeto de aplicação para o Gnome.

3 - Caixa de propriedades

Daqui é possível configurar os objetos da sua aplicação. Cor, Tamanho, Texto, posição etc. Além de definir também os sinais (eventos) a qual cada objeto responderá.

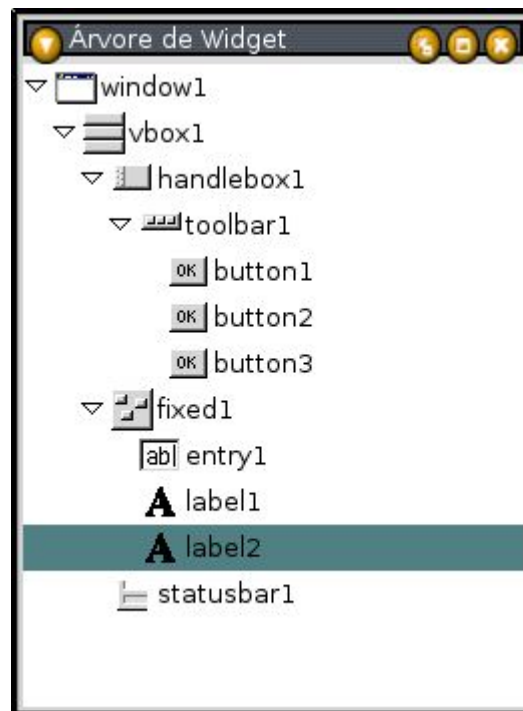


Na figura apresentada vemos as propriedades principais de um GtkEntry (uma caixa de entrada de texto).

4 - Árvore de Widgets

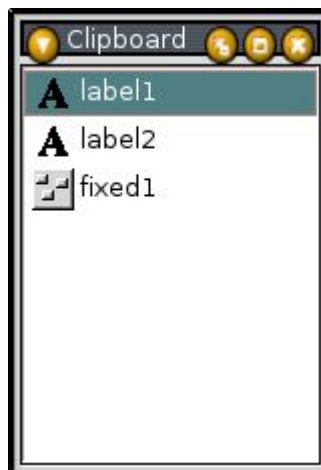
A partir da Árvore de widgets será possível ver a hierarquia de objetos da aplicação em desenvolvimento. Isso é útil na hora de selecionar um widget que se encontra inacessível através de um simples clique sobre ele, na janela da sua aplicação.

Este caso é muito comum de acontecer quando se deseja selecionar um container (ou frame), ou ainda uma janela, por exemplo, para realizar alguma operação sobre eles.



5 - Área de Transferência











Os objetos que forem copiados, recortados e colados estão aqui. Esta é uma área de transferência múltipla logo, poder-se-á ter vários objetos armazenados aqui para depois colá-los onde e como quiser.



























Objetos do GTK



O objetivo desta sessão não é apresentar todos os objetos do GTK e nem mesmo apresentar todos os oferecidos na paleta do Glade, já que isso pode ser obtido na documentação oficial do GTK encontrada em <http://lidn.sourceforge.net>

Abaixo apresentamos alguns componentes e um breve comentário sobre cada um.

<i>Objeto</i>	<i>Comentário</i>
 GtkWindow Janela	Qualquer aplicação que seja desenvolvida pra rodar em modo gráfico, seja em Windows®, ou X11 ou qualquer outro modo gráfico usa como elemento base uma janela. No GTK não poderia deixar de ser diferente.
 GtkMenuBar Barra de Menus	A barra de menus é o meio mais comum de se adicionar uma grande quantidade de funções em uma aplicação. A maioria das aplicações possuem uma barra de menus com pelo menos um menu. Ao inserirmos uma barra de menus no Glade ela já virá com alguns menus básicos (Arquivo, Editar, Exibir e Ajuda) e cada um deles com os seus comandos mais comuns.
 GtkToolBar Barra de Ferramentas	A barra de ferramentas é indispensável em programas profissionais e que possuem muitos comandos em menus. Aqui nós colocamos aqueles principais comandos que deverão ser acessados com maior frequência pelo usuário.
 GtkHandleBox Cabo de Caixa	Este é um recurso poderoso que nos permite arrastar o menu ou a barra de ferramentas para fora da área de trabalho da nossa janela de programa.
 GtkLabel Rótulo	O rótulo possibilita a exibição de mensagens, explicações e qualquer informação que deverá ser exibida dentro do programa e que não poderá ser alterada pelo usuário.
 GtkEntry Caixa de Entrada	Com a caixa de entrada o usuário poderá interagir com o nosso programa entrando com dados para manipulação. Por exemplo, com ele o usuário poderá entrar com os valores a serem usados em um cálculo, ou um nome, ou endereço de email que será cadastrado em uma agenda etc.
 GtkCombo Caixa de combinação	Como o nome diz, é uma combinação. Uma combinação de caixa de entrada com uma lista. Este componente permite que o usuário, tanto digite a sua opção, como possa escolher dentro de uma lista com itens já pré-definidos.
 GtkTextView Visualização de Texto	Este é um aperfeiçoamento no objeto GtkEntry que permite a visualização de grandes quantidades de texto. Enquanto o GtkEntry permite a inserção de pequenas quantidades de texto para entrar com pequenos dados no programa o GtkTextView permite a entrada de textos realmente longos ele é a base de qualquer editor de textos com base no GTK, como o Gedit, por exemplo
 GtkFixed Caixa de posicionamento livre	Esta caixa é um container que permite o posicionamento livre dos objetos assim como acontece em outras RADs lá no mundo Windows. Os objetos são posicionados com base nas coordenadas X e Y da janela. Isso é pouco comum de ser usado, pois apesar de simplificar a colocação dos objetos ele atrapalha na arrumação dos mesmos.
 GtkHBox Caixa horizontal	Este container permite que vários objetos fique lado a lado sem muito trabalho.

<i>Objeto</i>	<i>Comentário</i>
 GtkVBox Caixa vertical	Similar ao container anteriormente comentado, mas desta vez os objetos ficam dispostos um abaixo do outro, formando assim uma coluna de widgets
 GtkTable Caixa Tabular	Com este container é possível distribuir objetos uniformemente em toda a dimensão da janela. Ele facilita bastante a construção de telas de cadastro, onde poderíamos ter uma coluna com os rótulos e a segunda coluna com as caixas de texto correspondentes a cada campo.
 GtkStatusBar Barra de Status	Este componente permite-nos incluir uma barra de status no rodapé das janelas em nossos programas. Isso é interessante para mostrarmos informações sobre o que está acontecendo no processamento das tarefas. Todo programa profissional possui uma barra de status.
 GtkFrame Moldura	Com as molduras podemos organizar os objetos um uma janela com muitas opções. Isso é útil em telas de configuração, onde temos várias opções que poderão ser agrupadas, por exemplo uma moldura delimitando as opções de orientação de página, uma outra que delimite os tipos de papel etc.
 GtkHButtonBox Caixa de botões horizontais	Este componente é uma mão na roda para se criar caixas de diálogo, com ele podemos criar facilmente uma barra de botões de ação.
 GtkVButtonBox Caixa de botões verticais	Este é similar ao componente anterior, mas os botões são dispostos em coluna, de forma que fica um abaixo do outro. Geralmente usados para dispor os botões “OK”, “Cancelar” e “Ajuda” num dos cantos da sua tela.
 GtkVSeparator Separador Vertical	Assim como as molduras (GtkFrame) este componente serve apenas para organizar os objetos da janela. Ele simplesmente traça uma linha vertical separando sua janela em duas partes verticais
 GtkHSeparator Separador Horizontal	Similar ao widget anterior, mas este separa horizontalmente.
 GtkHPaned Divisória Horizontal	Divide a Janela em duas partes horizontalmente. Ele funciona como um container redimensionável diferentemente do GtkVSeparator este objeto divide e permite que estas partes sejam redimensionáveis através do cabo.
 GtkVPaned Divisória Vertical	Similar ao objeto anterior, mas divide em duas partes verticais.
 GtkRadioButton Botão de Seleção	Permite a seleção de opções. Seu uso é feito em conjunto com pelo menos dois destes organizados em um mesmo grupo, onde o usuário poderá selecionar uma dentre as várias opções disponíveis
 GtkToggleButton Botão de Alternação	Permite a ativação/desativação de opções em um programa. Por exemplo poderia ser usado para ativar ou desativar um cronômetro. Funciona como o “Botão de Verificação”.

<i>Objeto</i>	<i>Comentário</i>
 GtkCheckButton Botão de verificação	<p>Similar ao ao “Botão de Alternação” apresentado anteriormente. Mas geralmente é usado em conjunto com vários outros, apesar de não poder ser organizado em grupo.</p> <p>Usa-se por convenção o Botão de verificação, quando o usuário terá várias opções a marcar, ao contrário do Botão de Alternação que costuma-se usar quando há apenas uma opção a ser marcada.</p> <p>Este objeto inclusive pode assumir uma aparência similar a do widget anterior desativando a propriedade “Indicador”.</p>
 GtkImage Imagem	Este widget possibilita a visualização de imagens
 GtkDrawingArea Área de Pintura	Com este widget podemos criar aplicativos de desenho. Ao inserirmos um objeto destes o Glade automaticamente o colocará dentro de uma janela de rolagem possibilitando que a imagem área seja visualizada em toda sua dimensão, caso haja necessidade.
 GtkScrolledWindow Janela de Rolagem	Com este widget podemos “rolar” o conteúdo de um objeto muito grande que não caiba na janela. Ele é incluso automaticamente pelo Glade quando usamos os objetos “GtkDrawingArea”, “GtkTextView”, “GtkTreeView” e alguns outros.
 GtkTreeView Visão de Árvore ou lista	Este widget permite o desenvolvimento de aplicações que necessitem uma árvore hierarquica ou a listagem detalhada de alguma coisa. Seu uso pode ser notado em programas como Nautilus, Synaptic etc.
 GtkButton Botão	O botão padrão. Não há muito o que comentar sobre ele. É um dos componentes mais comuns em uma aplicação
 GtkOptionMenu Menu de Opções	O Menu de opções é uma alternativa a Caixa de combinação, com a diferença de que este não permite a edição do seu conteúdo pelo usuário.
 GtkSpinButton Botão Giratório	O botão giratório é similar a caixa de entrada (Entry) seu uso é exclusivo para entrada numérica, pois com ele o usuário tanto pode digitar, quanto escolher usando a “setinhas” do botão giratório.
 GtkViewPort Porta de Visualização	- a escrever -
 GtkNotebook Notebook	Com o “Notebook” podemos colocar muitos objetos em uma janelinha, simplesmente a dividindo em páginas.
 GtkDialog Caixa de diálogo	Uma caixa de diálogo padrão. Isso facilita o trabalho de criação de caixas de diálogo, pois do contrário teríamos que criar uma janela e personaliza-la manualmente com os botões e rótulos necessários.
 GtkFontDialog Diálogo de seleção de Fontes	A tradicional caixa para seleção de fontes, muito usada em programas que manipulem texto.

<i>Objeto</i>	<i>Comentário</i>
 GtkColorDialog Diálogo de Seleção de Cores	A caixa para seleção de cores
 GtkFileDialog Diálogo de Seleção de Arquivos	Caixa para seleção de arquivos. Costuma ser usada em programas onde se faz necessário abrir ou salvar arquivos.

Libglade

O Libglade é uma biblioteca que permite a utilização das interfaces geradas pelo Glade, como elas são, ou seja, ao salvar o projeto no Glade é gerado um arquivo XML com as descrições das janelas e seus objetos. O que o Libglade faz é ler este arquivo em tempo de execução de forma que as janelas não são convertidas em GTK e compiladas com o projeto.

Uma das grandes vantagens disso é que se for preciso mudar o layout da janela, não será necessário recompilar os fontes novamente, desde que não se mude o nome dos objetos e nem do arquivo ".Glade".

A grande desvantagem disso é que de alguma forma o seu programa deverá saber em tempo de execução onde está o arquivo ".glade" com as janelas. Uma forma fácil e simples de resolver isso é especificar no seu código o caminho completo para o arquivo de interface e que deverá ser transportado junto com o código.

O Glade por si só, como já citado antes permite gerar o código fonte em algumas linguagens. Mas depois que esse código começar a ser incrementado ele não conseguirá mais gerenciar aquele código e a partir daí será necessário fazer quase tudo sozinho, aí que entra a grande vantagem do libglade, se não quiser reescrever seu programa novamente a cada vez que for preciso mudar alguma propriedade dos objetos da aplicação, pois como o Glade vai recriar o arquivo ".glade" ele não tocará nos seus arquivos ".c".

Funções básicas

Pra que você comece a se habituar e a decorar alguns nomes: As funções básicas do GTK e do Libglade que vamos precisar pra começar nossos trabalhos são as seguintes:

```
gtk_init();  
gtk_main();  
glade_init();  
glade_xml_new();  
glade_xml_signal_autoconnect();  
glade_xml_get_widget();
```

Estas funções são primordiais e existirão em praticamente 100% dos programas baseados em GTK/Libglade

Função `gtk_init()`

Esta função inicializa e disponibiliza o GTK para sua aplicação. Ela deve ser a primeira a ser chamada por sua aplicação para que os objetos sejam criados adequadamente.

Exemplo:

```
gtk_init(&argc , &argv);
```

Em praticamente qualquer aplicação você a usará exatamente como neste exemplo. Note que geralmente os parâmetros passados a ela são referências aos parâmetros `argc` e `argv` passados a sua aplicação.

Função `gtk_main()`

Este será o “último” comando a ser chamado por sua aplicação no módulo principal, pois ele é o responsável por entrar no loop infinito que avalia os eventos que estão ocorrendo. Quando a sua aplicação avança deste comando, a interface gráfica é finalizada. Geralmente você irá querer fazer isso pra encerrar a aplicação.

Pra finalizar o seu programa você usará um comando apropriado que faz o GTK encerrar esse loop, chamado de “`gtk_main_quit()`”.

Função `Glade_init()`

Esta função é importante para inicializar a biblioteca Libglade. Mas ela tornou-se obsoleta e seu uso é desnecessário atualmente. Mas se a sua aplicação der problemas e você notar que o Libglade não foi inicializado então você poderá usar este comando logo após o `gtk_init()`.

Função `Glade_xml_new()`

Este talvez seja o comando mais importante pra construção de sua aplicação com o Libglade, pois é ele o responsável por ler a estrutura do arquivo XML gerado pelo Glade.

Este comando faz a leitura do arquivo de descrições XML contendo as janelas da sua aplicação e as carrega em um ponteiro do tipo `GladeXML`.

O protótipo deste comando é:

```
GladeXML* Glade_xml_new (const char * arquivo, const char *  
objeto_principal, const char * domínio);
```

Onde:

- Arquivo = Nome do arquivo “.Glade” contendo a sua interface
- Objeto_principal = Objeto inicial a partir de onde o Glade descerá na estrutura Xml pra construir sua janela. Isso é útil quando você possui a descrição de várias janelas no mesmo arquivo e só precisa montar uma delas. Você poderá deixar este parâmetro como `NULL` se quiser montar tudo, ou ainda se tiver apenas uma janela no seu arquivo.
- Domínio = Domínio para tradução do arquivo XML. Geralmente é deixado como `NULL` para

usar o default do sistema.

- O retorno é um novo objeto GladeXML ou NULL se falhar

Função `Glade_xml_signal_autoconnect()`

Tão importante quanto o comando `Glade_xml_new` temos este comando que é responsável pela conexão das funções a cada sinal (eventos) ocorridos em sua aplicação.

Protótipo:

```
void Glade_xml_signal_autoconnect (GladeXML *self);
```

Onde: “self” e a sua estrutura XML carregada pelo comando anteriormente comentado.

Se você der uma lida na documentação oficial do Libglade, verá que existem outras funções similares a essa e que permitem fazer a conexão de apenas alguns sinais em específico, mas os seus usos são apenas para algumas ocasiões especiais que estão fora do escopo deste material.

Função `Glade_xml_get_widget()`

Esta função, será a mais utilizada em todo o seu programa. Pois é com ela que você conseguirá associar um widget na sua interface XML com o respectivo ponteiro em tempo de execução. Você a usará pelo menos uma vez para cada objeto que precisar ler ou alterar suas propriedades.

Protótipo:

```
GtkWidget* Glade_xml_get_widget (GladeXML *self, const char *nome);
```

Onde:

- self = a sua estrutura XML já carregada na memória
- nome = nome do objeto que você quer associar (este deve ser o nome conforme esta definido no Glade).
- O retorno é um ponteiro do tipo GtkWidget que aponta pra região de memória onde esta o seu objeto.

Com isso já podemos partir para um primeiro programinha e começarmos a ver do que tudo isso é possível.

Hello World em libglade

Como de praxe em qualquer aprendizado de programação aqui vai o nosso “Hello World”. Este exemplo é até um pouco sofisticado demais para ser chamado assim, já que ele não vai mostrar apenas uma “janelica” escrita “Ola mundo!”, mas como será o primeiro projeto que faremos com o GTK/Libglade então esse nome será bem-vindo.

Nesse projeto, além da janela, ele terá rótulo e botões funcionais para que você possa ver como funciona as ligações de objetos a sinais, propriedades etc.

Em resumo todo o básico pra você começar. O motivo de usar este exemplo é pelo simples fato de que quando eu comecei com esta linguagem tive muito trabalho em pesquisar e ler vários textos e códigos fontes “perdidos” pela Internet (viva ao Google!). Todo exemplo que encontrava era só mostrando a colocar um “botãozinho”, ou um rótulo, compilar e pronto!

Espero portanto apresenta-lo algo mais do que um simples “Hello World”.

Uma aplicação desenvolvida em qualquer linguagem é feita em quatro etapas básicas:

- Desenho da interface – Onde montaremos a(s) janela(s) do nosso aplicativo de acordo com o necessário, incluindo os componentes mais adequados a cada entrada do usuário ou reposta que será apresentada a ele.
- Codificação – O trabalho mais difícil e demorado está aqui. Implementar o código fonte em si é sempre um trabalho demorado e cansativo, é aqui que surge 99% dos problemas em um sistema de informação.
- Compilação – Geralmente essa etapa se resume a um clique ou a digitação de algumas linhas de comandos no terminal – Este é o processo de “tradução” do que foi codificado em um linguagem de alto nível (como C /C++) para uma linguagem que o nosso sistema operacional possa entender
- Teste de funcionamento – Os testes devem ser feitos de forma bem prática e realistas de acordo com a necessidade de cada sistema – Se o seu sistema faz cálculos por exemplo, nada mais óbvio do que testar várias possibilidades de entradas de números diferentes, mas é imprescindível ver o que acontece se por engano o usuário entrar com alguma letra também.

Observem que estou me referindo as etapas de uma implementação de aplicação em modo gráfico. Aquelas partes de análise de sistema que todo programador detestou estudar na faculdade, você já deve estar cansado de saber que são necessárias, de serem feitas, antes do desenho da interface.

Os seguintes programas serão usados em cada etapa apresentadas:

Etapa 1 usaremos o Glade – Afinal ele é que é o foco do nosso trabalho

Etapa 2 usaremos o gedit – ou outro editor de sua preferência – o motivo da escolha deste editor é que ele foi feito em GTK (já que estamos usando o gnome – pelo menos eu estou usando – e por que ele faz destaque de sintaxe em C/C++). Vale aqui frisar também a existência do IDE “Anjuta” (<http://anjuta.sourceforge.net> – meu preferido) que é completíssimo no quesito integração de ferramentas, mas como vamos fazer um projeto simples, encerro aqui os comentários sobre ele até o momento oportuno.

Etapa 3 – Pra compilação o “todo-poderoso” g++ - Não haveria motivos que justificassem o uso de outro, já que estamos usando o GNU/Linux.

Etapa 4 – Pra teste de funcionamento a tradicional linha de comando do xterm ou outro terminal de sua preferência.

O motivo de usarmos estas ferramentas é que elas estão acessíveis em quase 100% das instalações Linux em desktop. Se a sua estiver faltando alguma delas você deverá providenciar imediatamente.

Começando

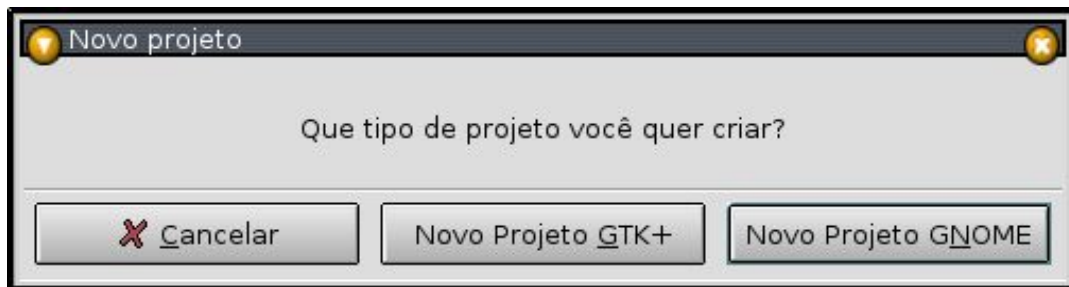
Antes de mais nada é recomendável que você tenha o bom costume de criar um diretório pra armazenar os seus projetos, afinal eles terão vários arquivos e isso vai te ajudar bastante a manter tudo organizado.

A partir da linha de comando você poderá emitir o comando abaixo para iniciar o Glade, ou ainda poderá iniciá-lo a partir do menu principal do seu ambiente de trabalho. No caso do Gnome será no menu **Aplicações** / menu **Desenvolvimento** / Opção **Glade-2**(ou **Construtor de Interfaces Glade**).

```
welrbraga@wrbfire:~$ Glade-2
```

Criando a interface com Glade

Com o Glade já aberto você irá criar um novo projeto, clicando no menu **PROJETO** e escolhendo o comando **NOVO**. Se você tiver instalado o Glade com suporte ao gnome será questionado quanto ao tipo de projeto, escolha novo projeto GTK.



A nossa interface será simples: Teremos uma janela, um rótulo e três botões (um pra mostrar uma mensagem, outro pra limpar a mensagem e mais um pra encerrar). Isso é mais do que suficiente pra um Hello World!

Criando a janela

Clique no botão “**Janela**” da paleta de componentes – Isso vai exibir uma janela vazia na sua tela.

A janela é o componente principal de uma aplicação que rode em modo gráfico, pois é nela que incluiremos os objetos que o usuário terá acesso para controlar a aplicação.

Ao contrário de outras linguagens do mundo Windows, como o Delphi e o C++ Builder [www.borland.com.br] Em Glade (e algumas outras linguagens) a janela se apresenta realmente vazia e se você inserir um objeto qualquer nela, este objeto ocupará toda a área cliente impedindo a inclusão de novos objetos. Por isso devemos sempre usar frames (ou containers) pra dividir a janela de acordo com o que queremos.

Dividindo a Janela pra incluir os objetos

Pra quem esta habituado com linguagens estilo Borland, como Delphi, Kylix C++ Builder. Este poderá ser um conceito estranho e meio complicado de se entender. Eu mesmo reclamei muito até descobrir os benefícios que isso traz.

Por exemplo, quantas vezes você já teve dor de cabeça pra colocar 5 rótulos exatamente um abaixo do outro e exatamente com o mesmo tamanho? Com o uso de frames bastaria por um frame do tipo “caixa vertical” e seus rótulos dentro dele. Tudo rápido, prático e o que é melhor: Grátis! Em outras

linguagens você precisaria selecionar todos eles e usar uma opção de alinhamento (se existir) ou definir na munheca a posição de cada um através da propriedade “canto esquerdo” deles, mas com um simples descuido com eles poderia dizer “tchau-tchau” arrumação.

Em Glade temos 4 tipos de frames:

- Caixa Horizontal – Dispõe objetos horizontalmente
- Caixa Vertical – Dispõe objetos verticalmente
- Tabela – Dispõe objetos em forma de tabela
- Posições Fixas – Permite a movimentação livre dos objetos (como no Delphi e C++ Builder citados anteriormente)

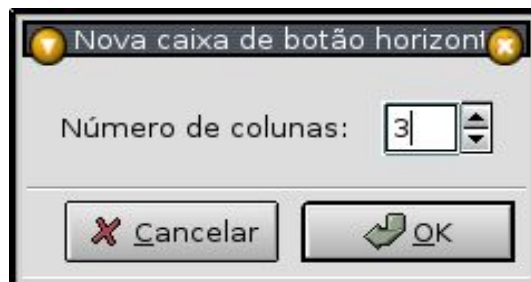
No nosso caso usaremos a “Caixa vertical” por permitir o uso de dois objetos um abaixo do outro. É claro que futuramente você poderá alterar isso, com um pouco de prática, mas no momento deixemos assim.

- Clique no botão “**Caixa Vertical**” [GtkVBox] e em seguida clique sobre a sua janela. Na pergunta que será exibida você vai escolher 2 linhas. Se você lembrar o que foi exposto anteriormente sobre como será a nossa aplicação, verá que usaremos um total de quatro objetos e no entanto estou reservando espaço para dois! O motivo disso é que ao invés de inserir os três botões, um de cada vez, usaremos uma caixa de botões que facilita bastante o nosso trabalho.



Os outros objetos

- Agora clique sobre o botão “**Rótulo**” [GtkLabel] na paleta de componentes e clique na parte superior do frame. Com isso você verá um rótulo escrito “label1” e a parte de baixo toda vazia.
- Clique agora em “**Caixa de Botões Horizontais**” [GtkHButtonBox]



Na janela exibida nós confirmaremos 3 colunas (ou três botões). Com isso a nossa janela deverá estar como a apresentada abaixo.



Se ela estiver com as proporções diferentes disso vá até a Janela Principal do Glade (aquela com botões de novo, abrir e salvar projeto), clique sobre o nome da janela “window1” e na caixa de propriedades, na ficha “widget” altere a propriedade *REDIMENSIONÁVEL* para *NÃO*.

Perfumaria!

Poderíamos deixar a janela como está para começarmos a programar, mas se você quiser deixá-la com uma cara mais amigável então precisaremos alterar algumas propriedades dos objetos. Como as alterações que faremos aqui são meramente “cosméticas” você pode pular essa seção, se quiser, e continuar a partir da próxima mas com isso poderá estar perdendo a grande chance de aprender como deixar as suas aplicações com cara de “aplicação de verdade”.

Vale uma dica importante aqui: Alguns objetos são complicados de selecionar. Se tiver dificuldade pra isso você poderá exibir a “árvore de widgets” (a partir do menu Ver) pra facilitar essa tarefa.

Janela

Vamos alterar as seguintes propriedades da janela do nosso projeto:

Na ficha Widget alteraremos:

<i>Propriedade</i>	<i>Descrições</i>	<i>Novo Valor</i>
Título	Título exibido na janela – Isso fara com que a nossa janela apresente este título, ao invés de “window1”, como e o padrão.	Hello World
Posição	Posição onde a janela será exibida na tela - O padrão e que ela seja exibida onde foi deixada na hora da ultima compilação. Com isso forçaremos a começar sempre centralizada na tela do usuário.	Center
Redimensionável	Se a janela poderá ser ou não redimensionada – não há necessidade para ser diferente de “não”, afinal esta e uma aplicação pequena e que o usuário não precisara de uma Área tão grande.	Não

Rótulo

Apenas uma propriedade será alterada aqui.

<i>Propriedade</i>	<i>Descrições</i>	<i>Novo Valor</i>
Rótulo	Define o texto exibido no rótulo – Nossa aplicação começara com o rotulo vazio, ou seja sem nada escrito.	[Apagar]

Botões

Apenas a propriedade “Botão de estoque” será alterada nestes objetos. Esta propriedade define automaticamente uma figura e um rótulo para o botão de acordo com o padrão do Gnome, facilitando bastante a vida do programador e deixando o nosso programinha com aparência mais profissional.

Você deverá clicar em cada um deles e mudar esta propriedade conforme abaixo:

<i>Objeto</i>	<i>Novo Valor</i>
Button1	Aplicar
Button2	Limpar
Button3	Sair

Com isso a ornamentação da janela está concluída. Poderemos agora definir o que será “vivo” no nosso programa, ou seja, definir a quais eventos o nosso programa deverá responder quando o usuário interagir com ele.

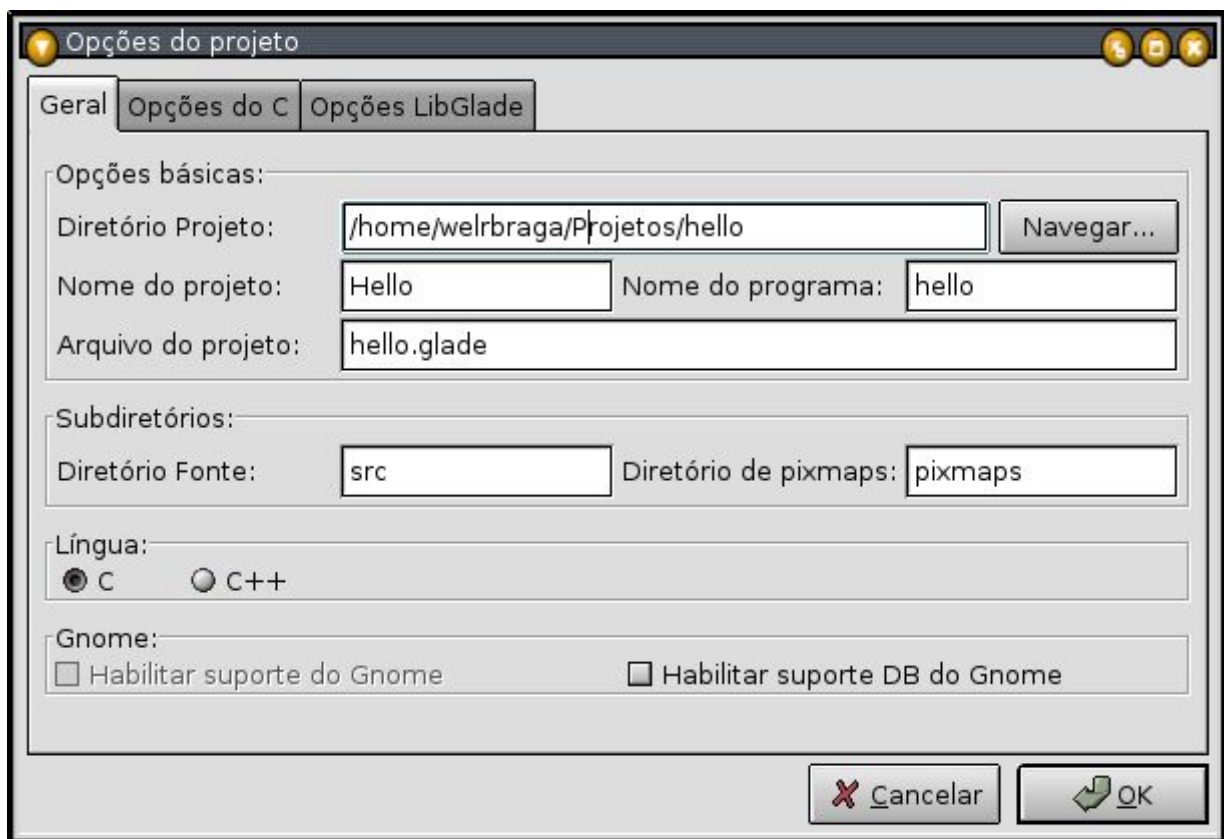


Mas antes disso vamos salvar a nossa interface, e testar como ela se comportará depois de tudo concluído.

Salvando a interface pela primeira vez

Para salvar a interface agiremos como em qualquer programa: clica-se no botão SALVAR, ou ainda o atalho CTRL+S ou a partir do menu PROJETO usa-se o comando SALVAR (A gosto do freguês).

Ao fazer isso a tela abaixo será apresentada e a única coisa que precisamos alterar é o nome do projeto, pois ele definirá automaticamente o diretório, nome do programa e arquivo do projeto, não precisamos definir a linguagem (aqui chamada de língua), pois não usaremos o Glade para gerar o código fonte. Da mesma forma também não ira nos interessar as outras fichas desta janela.



Ao clicar em OK o diretório hello será criado e dentro dele teremos dois arquivos: hello.Glade e hello.glade. Só tenha o cuidado de observar em que local o Glade está salvando sua interface, pois você irá pra lá via linha de comandos (geralmente ficará em ~/Projetos/seu_projeto.)

Como o diretório de projeto está criado pra salvar todo o projeto, nós entraremos nele e a partir da linha de comando vamos chamar o Gedit criando um arquivo chamado main.cpp (poderia ser qualquer outro nome, mas eu gosto desse nome, por deixar mais claro a sua finalidade e também por facilitar a utilização de alguns scripts personalizados que eu criei pra facilitar certas tarefas.

Simplesmente abra um terminal e digite:

```
welbraga@wrbfire:~$ cd Projetos/hello
welbraga@wrbfire:~/Projetos/hello$ gedit main.cpp &
```

Ao abrir o Gedit, talvez ele queira saber se você deseja criar um arquivo com o nome “main.cpp”. Confirme que sim pra continuar.

Com o Gedit aberto digite o código fonte do seu programa.

O código fonte básico e comentado

Geralmente um código fonte pra iniciar uma aplicação libglade/GTK se parece com o mostrado abaixo. É claro que poderá existir algumas diferenças para alguns casos, mas no geral isso funciona com 90% das aplicações, por isso eu costumo mantê-lo num “diretório de modelos” e copio sempre que necessário fazendo as devidas alterações.

```
#include <glade/glade.h> // Carrega os recursos do Glade
#include <gtk/gtk.h> //Carrega os recursos do GTK
GladeXML * xml; //Cria um ponteiro para uma estrutura GladeXML
int main(int argc, char *argv[]) //O Corpo principal do Programa
{
    gtk_init(&argc, &argv); //Inicializa o GTK
    glade_init(); //Inicializa o Glade – Isto geralmente é opcional
    xml = glade_xml_new("hello.glade", "window1", NULL); //Lê o arquivo com a
    estrutura XML da sua aplicação
    glade_xml_signal_autoconnect(xml); //Conecta todos os sinais aos seus
    manipuladores
    gtk_main(); // O Loop de eventos do GTK
    return 0;
}
```

O código está todo comentado e como ele é praticamente padronizado à todas as aplicações não há muito o que se falar sobre ele. A não ser a linha:

```
xml = glade_xml_new("hello.glade", "window1", NULL);
```

Esta é a única onde acontecerão mudanças de acordo com a necessidade do programa. Onde “hello.glade” é o nome do arquivo com a interface XML e “window1” é o nome da janela principal do programa. Vale lembrar que o nome do arquivo “.glade” deve ter o caminho completo especificado, pois do contrário você só conseguirá rodar a aplicação a partir do diretório onde ela está. Eu particularmente costumo copiar este arquivo para “/var/lib/”nome-do-programa” e especificar o caminho completo, pois desta forma ao criar um pacote “.deb” ou “.rpm” pra distribuí-lo vou ter garantia de que o programa irá achar corretamente onde está a interface, existem outras maneiras de se resolver esse problema também mas por hora vamos deixar assim e usaremos a linha de comandos para rodar o programa quando necessário.

As demais linhas do código fonte estão comentadas e você poderá voltar a seção “funções básicas” para obter mais detalhes.

Após digitar esse código (ou simplesmente copiar e colar) salve-o e voltemos para a linha de comando. Agora nós vamos compilar o nosso programa e verificar se ele está funcionando.

Compilando

A compilação do programa via linha de comando se resumirá em apenas uma linha como a mostrada abaixo:

```
g++ -o hello main.cpp `pkg-config --cflags --libs libglade-2.0`
```

Com isso nós estamos invocando o g++ pra gerar um executável chamado hello, a partir do arquivo fonte “main.cpp”.

Observe que estamos também chamando o pkg-config e passando a sua saída para o g++ (através do “-” - “crases”). De forma grosseira e objetiva o que este comando faz é analisar todas as dependências da libglade-2.0 e indicá-las ao compilador de forma que nós não tenhamos que digitar mais nada. Só

a título de curiosidade, se você não fizesse isso a sua linha de comando deveria ser a seguinte:

```
g++ -o hello main.cpp -I/usr/include/gtk-2.0 -I/usr/include/libxml2 -I/
usr/lib/gtk-2.0/include -I/usr/include/atk-1.0 -I/usr/include/pango-1.0 -I/
usr/include/freetype2 -I/usr/X11R6/include -I/usr/include/glib-2.0 -I/
usr/lib/glib-2.0/include -Wl,--export-dynamic -lglade-2.0 -lgtk-x11-2.0
-lxml2 -lpthread -lz -lgdk-x11-2.0 -latk-1.0 -lgdk_pixbuf-2.0 -lm
-lpangoxft-1.0 -lpango-1.0 -lpango-1.0 -lobject-2.0 -lmodule-2.0 -ldl
-lglib-2.0
```

Observe que há uma diferença respeitosa em usar ou não este comando. Portanto use o primeiro comando no seu terminal e vamos compilar o nosso programa.

Cerca de uns cinco a trinta segundos depois (dependendo a velocidade do seu equipamento) o terminal deverá exibir a linha de comando novamente e sem emitir mensagem alguma. Se isso ocorreu então a aplicação foi compilada com sucesso.

Use o comando 'ls -l' para ver se há um arquivo executável no seu diretório e tiver vamos rodá-lo.

```
welbraga@wrbfire:~/Projetos/hello$ g++ -o hello main.cpp `pkg-
config --cflags --libs libglade-2.0`
welbraga@wrbfire:~/Projetos/hello$ ls -l
total 36
-rwxr-xr-x  1 welbraga welbraga  13238 2003-12-14 18:02 hello
-rw-r--r--  1 welbraga welbraga    3181 2003-12-14 17:38
hello.Glade
-rw-r--r--  1 welbraga welbraga    271 2003-12-14 17:38
hello.gladep
-rw-r--r--  1 welbraga welbraga    592 2003-12-14 18:02
main.cpp
-rw-r--r--  1 welbraga welbraga    592 2003-12-14 18:01
main.cpp~
welbraga@wrbfire:~/Projetos/hello$ ./hello
```

Ao rodar o programa a janela que criamos deverá aparecer na tela conforme nós a programamos.

Só observe que ao clicar nos botões da janela ela não responderá pois não definimos evento algum nela.

Note que nem mesmo o botão fechar da barra título está funcionando. Se você clicar nele a janela se fechará, mas o programa continua rodando (veja que a linha de comando não apareceu no terminal). Use CTRL+C para quebrar a aplicação e na próxima etapa vamos corrigir este problema.

Ate aqui meus parabéns seu programa com Glade e GTK já começou a dar sinal de vida.

Se você achou complicado chegar ate aqui, não desanime, a primeira vista parece que é mesmo, mas com perseverança e força de vontade você habituará e logo verá que não é tão complicado assim. Mas acredito que se você chegou até aqui é porque quer aprender e está disposto a ler e escrever muito, afinal de contas já deve estar acostumado com esta característica do C/C++.

Dando vida ao programa

Antes de avançar deveremos ter dois conceitos em mente logo antes de começarmos:

1. Função manipuladora é a função que dá “vida” a um evento. É a função onde programaremos a finalidade de determinado objeto.
2. Sinal é o acontecimento que será interceptado e que acionará a nossa função manipuladora.

O que devemos fazer agora é voltar ao Glade e definir quais serão os sinais que a nossa aplicação responderá e qual será a função manipuladora que estará associada a eles.

Você deve se lembrar que ao rodarmos a aplicação e clicarmos no botão fechar da barra de título a janela se fechava, mas a aplicação continuava rodando. Isso ocorreu porque a nossa aplicação entrou em um loop infinito “`gtk_main()`” e em momento algum houve um comando que o fizesse sair desse loop. Pois bem, selecione a janela principal (use a Árvore de Widgets, se for necessário) e na janela de propriedades vá até a ficha de Sinais.

Na lista de sinais você deverá clicar no botão [...] e selecionar o sinal chamado “`destroy`” (O penúltimo evento – Este evento ocorre sempre que um objeto é destruído) ao confirmar o Glade vai sugerir um manipulador chamado “`on_window1_destroy`”. Mas esse nome é muito grande e feio, vamos apagar isso e simplesmente digitar “`fechar`”. Clique no botão “Adicionar” e pronto. Dessa forma ao invés de criarmos uma função com o nome “`on_window1_destroy`” simplesmente criaremos com o nome “`fechar`”.

Agora ative o sinal “`clicked`” do botão “Aplicar”. Para isso clique no primeiro deles e repita o mesmo procedimento acima, mas dessa vez escolhendo o sinal “`clicked`” e o manipulador altere para “`aplicar`” e não esqueça que no final você deve clicar o botão “Adicionar”, pois do contrario ele não aceitará as alterações que foram feitas.

Repita o mesmo procedimento com os outros dois botões. Com isso o nosso programa deverá responder aos seguintes sinais:

<i>Widget</i>	<i>Sinal</i>	<i>Manipulador</i>
window1	destroy	fechar
button1 “Aplicar”	clicked	aplicar
button2 “Limpar”	clicked	limpar
button3 “Sair”	clicked	fechar

Observe que o botão “button3” responderá ao sinal “`clicked`” usando o mesmo manipulador do sinal “`destroy`” da nossa janela, afinal de contas a função deles são as mesmas e isso vai nos economizar um manipulador e um pouco de trabalho.

Lembre-se que o C diferencia maiúsculas e minúsculas, portanto tenha cuidado ao digitar.

Após salvar as alterações feitas no Glade, se rodarmos a aplicação novamente (não precisa

recompilar – lembre-se que o libglade interpreta a estrutura “.glade” em run-time) verá que ele apresentará as seguintes mensagens no terminal:

```
welbraga@wrbfire:~/tutorial-Glade/hello$ ./hello
(hello:5034): libglade-WARNING **: could not find signal handler
'fechar'.
(hello:5034): libglade-WARNING **: could not find signal handler
'aplicar'.
(hello:5034): libglade-WARNING **: could not find signal handler
'limpar'.
```

Observe que o libGlade já está tentando associar os sinais aos respectivos manipuladores, e isso sem que precisássemos recompilar todo o projeto. Se nós já tivéssemos programado estes três eventos a aplicação estaria funcionando e não precisaríamos fazer mais nada a não ser aplaudir. Como não o fizemos deveremos agora programar as três funções.

Então voltemos ao gedit e acrescentaremos as seguintes linhas no nosso programa:

Protótipo dos Manipuladores Eventos

As linhas abaixo são os protótipos das nossas funções manipuladoras e por isso deverão estar no começo do nosso programa logo acima da função main() .

```
extern "C" {
    void fechar(); //Fechar a aplicação
    void aplicar(); //Aplicar
    void limpar(); //Limpar
}
```

Vale destacar aqui que os protótipos estão dentro da sessão “extern C” por que o GTK feito em C e por isso exige que os sinais sejam linkados como C padrão. Por estarmos usando o G++, que é um compilador pra C++, devemos forçá-lo a usar este tipo de linkagem. Se estivéssemos usando o GCC e compilando com o C padrão isso não se faria necessário, mas perderíamos todo o poder que a linguagem C++ nos oferecerá futuramente.

As funções manipuladoras

O que mais dá trabalho pra trabalhar com o GTK é o acesso aos widgets da sua aplicação. Você não pode acessá-los diretamente e ainda precisa criar os ponteiros que referencie exatamente quem você esta querendo usar. Mas com o costume você verá que isso não e nenhum bicho de sete cabeças.

Função Fechar()

Como já comentado antes a função fechar será acionada por dois sinais distintos: O sinal “destroy” emitido pela janela ao clicarmos no botão “fechar” convencional e pelo sinal “clicked” emitido pelo botão “button3”.

Já que ambos os sinais deverão responder da mesma forma, poderemos com isso economizar a digitação de código para uma função.

Observe abaixo que a programação de um sinal pra encerrar uma aplicação GTK consiste em usar um simples comando que é o “gtk_main_quit()”.

Você deve se perguntar: Eu poderia usar o “gtk_main_quit” lá na definição de função manipuladora do botão “Fechar” a partir do glade?

A resposta é sim e é não!!

Sim porque teoricamente funciona e não porque na pratica geralmente você fará algo mais antes de fechar a sua aplicação, como verificar se o arquivo está salvo, ou fechar as conexões com sockets, ou simplesmente pedir a confirmação do usuário.

```
void fechar() //Fechar a aplicação
{
    gtk_main_quit(); //Encerra o loop principal do GTK
}
```

Função aplicar()

A função aplicar também é uma função bem simples e cuja finalidade é simplesmente exibir a frase “Olá Mundo!”. Esta função será interceptada pelo sinal “clicked” do botão “button1” e cada vez que for chamada fará as seguintes tarefas na seqüência:

1 - Reserva um ponteiro do tipo GtkWidget que será usado para associar os dados do rótulo usado em nosso programa. Isso é importante, pois até então os nossos objetos são “meramente ilustrativos”, ou seja, não temos um meio direto de acessá-los e será a partir desse ponteiro que o acessaremos futuramente.

Este ponteiro poderia ser do tipo GtkLabel (o tipo que realmente estamos precisando), mas isso inverteria algumas coisas que veremos adiante.

2 - Usando o função “glade_xml_get_widget” faremos a associação do objeto “label1” ao ponteiro “mensagem” que declaramos anteriormente.

O que esta função faz é simplesmente ler a nossa estrutura XML e associar o objeto desejado ao nosso ponteiro do tipo GtkWidget

Observe que ela recebe dois valores: A estrutura XML (Que já fizemos a leitura logo no início da aplicação) e o nome do widget que será associado ao ponteiro.

Observe ainda que o nosso ponteiro não precisa ter o mesmo nome do objeto. Nesse caso o ponteiro se chama “mensagem” e o objeto associado a ele chama-se “label1”

3 - Finalmente Definiremos o texto do nosso label como sendo “Olá Mundo!” através da função “gtk_label_set_text()”.

Esta função requer como parâmetros um ponteiro para um tipo GtkLabel e o texto a ser atribuído.

Observe que a função “glade_xml_get_widget” retorna um ponteiro do tipo “GtkWidget” e

a nossa função “`gtk_label_set_text`” requer um ponteiro do tipo “`GtkLabel`” para resolver esse problema foi necessário fazer um “casting” para converter o `GtkWidget` “`mensagem`” em um `GtkLabel`. Sem este “casting” teríamos um erro em tempo de compilação.

```
void aplicar() //Exibe a mensagem definida
{
    GtkWidget * mensagem; // Cria um ponteiro para o widget
    mensagem = glade_xml_get_widget(xml, "label1"); //Associa o widget ao
    ponteiro
    gtk_label_set_text(GTK_LABEL(mensagem), "Olá mundo!"); //Define o texto que
    será exibido no widget
}
```

Função `Limpar()`

Esta função dispensa comentários. Se compararmos com a função `aplicar()` apresentada anteriormente veremos que ambas são idênticas. A diferença entre elas se dá apenas no detalhe que aquela primeira atribui o texto “Olá Mundo!” ao nosso rótulo, enquanto que esta última atribui um texto vazio “” de forma a limpar o nosso rótulo.

```
void limpar() //Limpa a mensagem
{
    GtkWidget * mensagem;
    mensagem = glade_xml_get_widget(xml, "label1");
    gtk_label_set_text(GTK_LABEL(mensagem), "");
}
```

Após digitarmos as três funções logo abaixo da nossa função `main()` poderemos recompilar o nosso programa e ao rodá-lo ela deverá estar 100% funcional e não retornar nenhum erro na saída do console.

Parabéns a sua primeira aplicação já funciona completamente!

Conclusão

Esta foi uma aplicação bem simples mas que demonstra como funciona o Glade. Ela poderia ser melhorada, otimizada, refinada etc. Mas para um simples “Hello World” creio que está de bom tamanho.

Sugiro a você como exercício retirar a redundância das funções “`aplicar`” e “`limpar`”. Existem várias maneiras de se fazer isso, mas fica a seu critério a forma como proceder, afinal de contas programar é uma questão de gosto.

APENDICE I - Simplificando as listas encadeadas com a GList

Quem já estudou listas encadeadas em C/C++ e não resmungou do trabalho que dá usar este poderoso mas complicado recurso da linguagem que me atire a primeira pedra.

As listas encadeadas são recursos extramente importantes no desenvolvimento de aplicações em C e são usadas como uma espécie de “vetor” dinâmico, onde os elementos são ligados uns aos outros formando uma seqüência.

Existem dois tipos básicos de lista encadeada: A lista encadeada simples e a lista duplamente encadeada. A diferença entre ambas é que a primeira só pode ser percorrida de frente pra trás, uma vez que cada elemento aponta apenas para o seu elemento seguinte; enquanto que a lista duplamente encadeada pode ser percorrida em ambos os sentidos uma vez que cada elemento possui referência ao seu antecessor e ao seu sucessor.

Cada um deste tipo de lista pode ser usado para se criar listas circulares, filas, pilhas, árvores etc. Mas isso está fora do escopo deste material e caso esteja interessado neste assunto você encontrará um farto material sobre isso apenas buscando no nosso bom amigo “google” (www.google.com.br) futuramente talvez eu venha a abordar esses assuntos em outro artigo, e até mesmo se você, caro leitor, se tiver conhecimento sobre o assunto e quiser contribuir estamos a sua disposição.

O desenho abaixo representa um elemento simples de uma lista encadeada

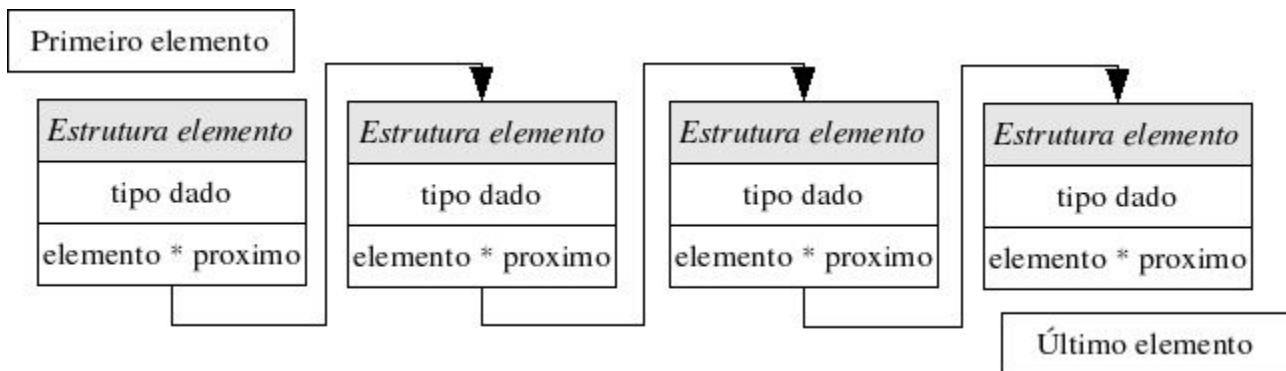


Figura1: lista encadeada simples

Note que cada elemento possui um ponteiro que aponta para o elemento seguinte da estrutura.

O objetivo deste artigo é mostrar como trabalhar com listas duplamente encadeadas, por elas terem uma quantidade de aplicações maior já que, como foi dito anteriormente, estas podem ser percorridas em dois sentidos. Mas se por algum motivo você precisar da lista encadeada simples isso poderá ser implementado facilmente com o tipo “GSLlist” que também está na biblioteca Glib.

Se você é daqueles que precisam de um bom motivo pra estudar, então vou lhe dar apenas um bom motivo para entender a lista encadeada: Ao fazer um programa com o Glade que use o objeto “gtkcombo” a lista de itens que são exibidos neste componente deve ser do tipo “GList” (que é uma lista duplamente encadeada!).

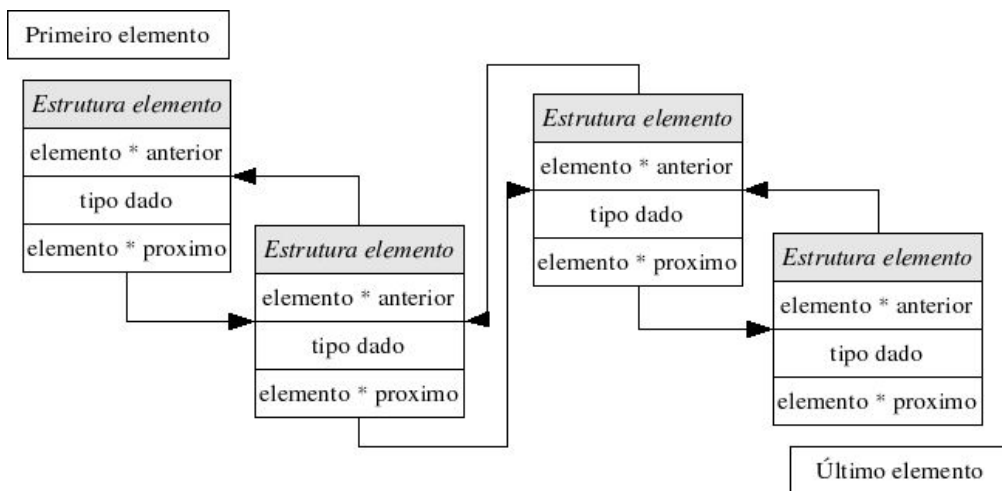


Figura 2: Lista duplamente encadeada

Observe no desenho da lista duplamente encadeada que cada elemento possui um ponteiro para o elemento seguinte, como na lista simples, mas possui também um outro ponteiro para o elemento anterior.

Listas duplamente encadeadas em C

Se ainda não ficou claro é bom notar que em uma lista encadeada cada elemento usa ponteiros para referenciar os outros elementos e como tudo deve ocorrer na dinamicamente precisamos usar as tradicionais funções “malloc” e “free” no C ANSI ou ainda as funções “new” e “delete” no C++.

Tipicamente uma lista encadeada para armazenar strings ficaria como segue:

```
struct s_strings {
    char *texto;
    s_strings *proxima;
    s_strings *anterior;
};
```

Observe que na prática nada mais temos do que uma estrutura simples contendo o dado principal que estamos querendo armazenar (nesse caso um ponteiro para “char*”) e duas referências ao tipo “s_string” que apontarão para a estrutura anterior e para a seguinte.

Para o leitor que tem dificuldades em trabalhar com um único ponteiro e nunca trabalhou com listas, imagine lidar com todos esses ponteiros de uma só vez!

É claro que exercitando adquiri-se certa prática e habilidade, mas eu confesso que dei graças a Deus quando descobri o tipo GList da Glib. Portanto não vou me esticar mais mostrando como usar a lista mostrada acima, mas sim com a nossa Glib.

O tipo GList

A Glib implementa o tipo GList como sendo uma estrutura de lista duplamente encadeada nos moldes da estrutura já mostrada.

Olhando a documentação desta biblioteca (eu particularmente faço isso usando o programa “Devhelp” que é um navegador de documentação) podemos ver a estrutura que é exatamente o mostrado abaixo:

```
struct GList {
    gpointer data;
    GList *next;
    GList *prev;
};
```

Compare as duas estruturas e veja a semelhança. Observe ainda que o campo “data” é do tipo “gpointer”.

```
typedef void *gpointer;
```

Na realidade o `gpointer` é um ponteiro do tipo “void”, ou se você preferir um ponteiro “não-tipado” o que significa que o seu campo “data” poderá assumir qualquer tipo de dado, seja ele um dado simples como um “int” ou até mesmo uma complexa estrutura de dados.

Vale ainda ressaltar que em alguns momentos você verá o tipo “gconstpointer” que é representado como está abaixo:

```
typedef const void *gconstpointer;
```

Assim como “gpointer” este é um ponteiro para o tipo “void”, sendo que desta vez ele aponta para um dado constante. Isto geralmente é usado em protótipo de funções para que o dado passado a elas não seja alterado.

Para criarmos uma lista do tipo `GList` é necessário tão somente o código abaixo:

```
GList *minha_lista = NULL;
```

Observe que a lista deve ser iniciada como `NULL` para que não tenhamos problemas futuros. Assim ela é considerada vazia.

Funções Típicas para manipular a glist

Se considerarmos uma lista como um tipo de cadastro (um cadastro de endereços, por exemplo), as principais ações que são feitas nesse cadastro são: inclusão, remoção, classificação, procura e listagem. Veja então as funções que usamos pra isso.

Funções de inclusão

As funções a seguir são usadas para inclusão de dados na glist:

g_list_append()

Usada para adicionar um novo elemento no final da lista.

```
GList* g_list_append (GList *list, gpointer data);
```

g_list_prepend()

Usada para adicionar um novo elemento do início da lista.

```
GList* g_list_prepend (GList *list, gpointer data);
```

g_list_insert()

Usada para adicionar um novo elemento em determinada posição.

```
GList* g_list_insert (GList *list, gpointer data, gint position);
```

g_list_insert_sorted()

Usada para inserir elementos ordenadamente na lista.

```
GList* g_list_insert_sorted (GList *list, gpointer data, GCompareFunc func);
```

Observe que a função `g_list_insert_sorted()` precisa receber o endereço de uma função de comparação, definida pelo programador previamente e seguindo algumas regras que vamos comentar adiante.

Funções de remoção

As funções a seguir servem para remover elementos da nossa lista.

g_list_remove()

Use-a para remover um único elemento da lista. Para usar esta função basta apenas informar o nome da lista e o conteúdo do elemento a ser removido.

```
GList* g_list_remove (GList *list, gconstpointer data);
```

É importante destacar que se existir dois elementos com o mesmo conteúdo, apenas o primeiro será removido. Se você desejar remover todos os elementos com este mesmo conteúdo use a função “`g_list_remove_all()`” que possui a mesma sintaxe.

g_list_free()

Esta função é usada para liberar a memória alocada pela sua estrutura e deve ser chamada explicitamente em seu programa quando tiver terminado com o uso da estrutura `GList` em questão.

```
void g_list_free (GList *list);
```

Outras funções

Seria uma perda de tempo ficar escrevendo cada função aqui e explicar o que já está explicado. Todas as funções da `glib` estão documentadas e podem ser consultadas diretamente no site oficial www.gtk.org, ou no site LiDN <http://lidn.sourceforge.net> que possui a documentação de praticamente todas as bibliotecas e ferramentas que você precisa e até mesmo localmente consultando os seus diretórios `/usr/share/doc`, `/usr/doc` e `/usr/local/doc` (Use um navegador de documentação como o “`Devhelp`”) para facilitar a sua vida.

Segue abaixo um breve resumo de mais algumas funções apenas para aguçar a sua curiosidade de ler a documentação.

- `g_list_find()` - Localiza um elemento com o dado especificado
- `g_list_length()` - Retorna o número de elementos na sua lista
- `g_list_index()` - Retorna a posição do elemento que contém o dado especificado
- `g_list_foreach()` - Executa uma determinada função para cada elemento da sua lista
- `g_list_concat()` - Junta duas listas
- `g_list_sort()` - Ordena uma lista de acordo com as características da função de ordenação especificada
- `g_list_reverse()` - Inverte a lista (o primeiro elemento torna-se o último, o segundo torna-se penúltimo etc)

A GList na prática

Chega de teoria e vamos ao que todos querem: Por em prática.

Pra começar vamos abrir um terminal e criar uma pasta para armazenar os arquivos deste programinha de teste.

Lembrando que para criarmos uma pasta, após ter aberto a janela de terminal basta emitir o comando:

```
$ mkdir nome-da-pasta [ENTER]
```

Eu particularmente gosto de armazenar estes testes dentro de uma pasta chamada “Projetos” então o que eu faço é entrar nesta pasta e depois criar a subpasta para o programa de teste:

```
$ cd Projetos [ENTER]
$ mkdir testeglist [ENTER]
```

Agora é só rodar o seu editor de textos preferido. Eu aconselho que este editor faça a numeração de linhas, para facilitar o trabalho de depuração quando for necessário. (particularmente eu prefiro o Gedit).

```
$ gedit main.cc & [ENTER]
```

Com isso estamos rodando o gedit pronto para criar o arquivo principal do nosso teste e deixando o terminal livre para nobres tarefas, como rodar o gcc por exemplo.

Vamos começar digitando o seguinte código:

```
/*
 * Teste da GList
 * para compilar use:
 * gcc main.c -o testeglist `pkg-config --cflags --libs glib-2.0`
 */

//Inclusao da biblioteca glib
//isso e importante para usar o tipo GList que é o foco do nosso estudo
#include <glib.h>

//Criacao da Lista encadeada
GList * MinhaLista;

//Funcao principal do nosso programa de teste
int main (char argc, char **argv) {
    g_print("Teste da GList\n");
    return 0;
}
```

Até aqui, se compilarmos o nosso programa ele não deverá acusar qualquer erro, mas se for executado, também não deverá fazer nada de interessante.

Para não termos que nos estressar com os métodos de entrada do C (aqueles “scanf” da vida) vamos aproveitar o vetor de parâmetros do nosso programa e entrar com os itens da lista via linha de comandos.

Para isso inclua o código abaixo a partir da linha 14 (onde está o “return 0”).

```
gint i = 0; //Variável inteira de uso geral
MinhaLista = NULL; //É importantíssimo que a GList seja iniciada com NULL
```

Veja que nós somente criamos uma variável do tipo inteiro que será usada para diversos fins e o mais importante que é iniciar a GList com o valor NULL. Lembre-se que uma lista encadeada é um ponteiro e todos os ponteiros devem ser sempre iniciados para não termos dor de cabeça futuramente.

Inclusão de elementos na lista

Agora inclua o código seguinte logo acima do “return 0;” (linha 16).

```
if (argc > 1) { //Verifica se foi passado algum argumento
    for (i = 1 ; i < argc ; i++ ) { //Passa por todos os elementos da lista
        //Adiciona cada um dos elementos no final da lista
        MinhaLista = g_list_append(MinhaLista, (gpointer) argv[i]);
    } //for i
} //if argc
g_list_free(MinhaLista); //Libera o espaço alocado pela "MinhaLista"
```

Aqui a coisa começou a ficar interessante: primeiramente estamos verificando se foi passado algum parâmetro para o nosso programa (verificando o valor de “argc”). Se houver pelo menos um item (maior que 1, pois o nome do executável é o primeiro parâmetro) então começamos um loop do primeiro até o último e adicionamos cada um deles na nossa GList usando a função “append” que inclui um elemento no final da lista.

Observe que os nossos itens são do tipo “char*”, mas esta função requer um ponteiro “não-tipado” (“void*” ou “gpointer”), daí o motivo do “casting” que foi feito.

Outra coisa a frisar é a função “g_list_free()” antes de encerrar o programa. Isso vai liberar a memória usada pela nossa GList sem precisarmos nos preocupar de remover item por item na lista.

Quantos itens há na lista?

Se compilarmos o nosso programa e o rodarmos ele não deverá acusar nenhum erro, mas nós ainda não podemos saber o que está acontecendo com ele, pois apesar de estarmos inserindo itens na lista nós não temos nenhuma informação a respeito disso.

Insira a partir da linha 21 (“} //if argc”) o seguinte código:

```
//Mostra o total de elementos na glist
g_print("[Total de elementos: %d]\n", g_list_length(MinhaLista));
```

Com essa linha logo após o laço “for” nós poderemos ver a quantidade de elementos que foram inseridos na lista.

O comando “g_print” é o responsável por exibir na tela a mensagem informando o tamanho (sua sintaxe é a mesma do já conhecido “printf” do C) e o comando “g_list_length” retorna a quantidade de elementos na lista, precisando apenas receber a lista como parâmetro.

Agora compile o programa e rode. A saída deverá ser algo como o mostrado abaixo:

```
$ gcc main.c -o testeglist `pkg-config --cflags --libs glib-2.0`
$ ./testeglist GTK-Brasil www.gtk.cjb.net
Teste da GList
[Total de elementos: 2]
```

Emitindo um aviso em caso de falha

Observe que executamos o testeglist com dois argumentos e ambos foram adicionados à lista. Experimente rodar o programa sem argumentos e você verá que não teremos nenhuma mensagem informando o que está acontecendo não é uma boa pratica de programação deixar o usuário sem informação portanto vamos substituir a linha 23 (“} //if argc”) pelo código a seguir:

```
} else g_warning("Nenhum item foi passado para lista. A lista esta vazia");
```

Se o programa for compilado e executado agora, sem parâmetros teremos a seguinte saída:

```
$ gcc main.c -o testeglist `pkg-config --cflags --libs glib-2.0`
$ ./testeglist
Teste da GList

** (process:4650): WARNING **: Nenhum item foi passado para lista. A lista esta vazia
```

Veja que usamos o “g_warning()” ao invés da “g_print” para apresentar a mensagem de aviso. Isso assegura que esta mensagem seja passada para o console de erros (/dev/stderr ou outro que seja definido).

É interessante nós fazermos isso nos nossos programas para que pódamos filtrar as mensagens de erro quando necessário. Se você se interessou por este recurso veja o manual da glib e procure a respeito das funções (“g_print()”, “g_message()”, “g_warning”, “g_critical()” e “g_error()”).

Exibindo o que há na lista

Até aqui nosso programa já armazena elementos na lista e informa quanto foram armazenados, mas ainda não sabemos quais foram armazenados. Não existe uma função que apresente todo o conteúdo da lista, mas para resolver esse problema e mostrar o conteúdo da nossa lista usaremos a função “g_list_foreach()” que executa uma função criada pelo usuário para cada elemento da lista. Mas antes que isso seja possível precisamos definir o protótipo desta função de exibição que deverá se parecer com o protótipo abaixo:

```
void (*GFunc) (gpointer data, gpointer user_data);
```

Onde:

- (*GFunc) - é o nome da função,
- data - é o elemento que será passado pela função “g_list_foreach()” (no nosso caso será um “gchar”)
- user_data - pode ser um vetor de parâmetros extras para configurar a sua função.

Começaremos inserindo o protótipo a seguir antes da função “main” (linha 11):

```
void mostra (gpointer item, gpointer meusdados); //Mostra o item da lista
```

e o corpo da função listado abaixo no final do arquivo, após o “}” que fecha a nossa função “main” (linha 28):

```
void mostra (gpointer item, gpointer meusdados){ //Mostra o item da lista
    gchar *texto = (gchar *) item;

    //Executa o que foi prometido ;- )
    g_print(" * %s\n", texto);
}
```

A primeira linha útil desta função apenas cria uma variável onde o texto de cada elemento da nossa lista ficará armazenado e atribui a ela o item atual já convertido para “gchar *”.

Em seguida ele mostra esse texto usando a função “g_print()”.

Não é preciso dizer que apesar de termos criado a função ela ainda não é chamada em ponto algum do nosso programa, portanto antes de compilarmos o nosso programinha nós vamos até a linha 24 do nosso programa (entre o g_print(“[Total...”) e o } else g_warning()) e acrescentaremos a seguinte linha:


```
g_list_foreach(MinhaLista, mostra, "");
```

Como já dito antes a função “g_list_foreach” executa uma determinada função para cada elemento contido na lista. Nós a usamos para executar a função “mostra” que havíamos criado e pronto. Observe que no final foi passado “” para o parâmetro “meusdados” que pra esse caso não tem utilidade alguma, mas deve ser passado, nós veremos um exemplo prático de uso deste parâmetro mais adiante quando vermos a remoção de elementos.

Finalmente compile e rode o programa.

```
$ gcc main.c -o testeglist `pkg-config --cflags --libs glib-2.0`
$ ./testeglist Ana Jose Maria Teresa
Teste da GList
[Total de elementos: 4]
* Ana
* Jose
* Maria
* Teresa
```

Ordenando a lista

Para ordenar a lista nós usamos a função “g_list_sort()” ou a “função g_list_sort_with_data()”. Ambas precisam de uma função auxiliar que determina o critério de ordenação. A diferença que existe entre elas é que a segunda aceita parâmetros adicionais que serão enviados para a sua função de ordenação.

Veja abaixo os seus protótipos:

```
GList* g_list_sort (GList *list, GCompareFunc compare_func);
```

e

```
GList* g_list_sort_with_data (GList *list, GCompareDataFunc compare_func,
gpointer user_data);
```

Ambas retornam um ponteiro para o início da lista já ordenada e seus parâmetros são:

- list – A lista a ser ordenada
- compare_func – A função que define o critério de ordenação
- user_data – Dados que serão passados a função “compare_func” (só na função “g_list_sort_with_data”)

O protótipo da função “compare_func” para a a função “g_list_sort” é o que segue:

```
gint (*GcompareFunc) (gconstpointer a, gconstpointer b);
```

E para a função “g_list_sort_with_data” é:

```
gint (*GcompareDataFunc) (gconstpointer a, gconstpointer b, gpointer
user_data);
```

Ambas recebem dois itens que serão comparados (enviados pelas funções “g_list_sort” e “g_list_sort_with_data”) e a última recebe ainda algum argumento que seja necessário passar para sua função.

Ao criar estas funções você deve considerar que os valores de retorno devem seguir a regra:

- Valor negativo se $a < b$;
- 0 se $a = b$;
- valor positivo se $a > b$.

Ex.:

Considere uma lista contendo os itens: M R B M

Ao comparar “M” com “R” deve ser retornado valor positivo (5), pois “M” é menor que “R”

Ao comparar “M” com “B” deve ser retornado valor negativo (-11), pois “M” é maior que “B”

Ao comparar “M” com “M” deve ser retornado valor 0 (zero) , pois “M” é igual a “M”

Pra quem ainda não entendeu o porque disso, considere a tabela ASCII onde cada caracter corresponde a um número de 0 a 255. “M” vale 77, “R” vale 82 e “B” vale 66. O que o computador faz é subtrair os valores ASC do primeiro com o segundo e retornar a diferença. Você pode ver essa tabela com o programinha abaixo:

```
#include <glib.h>

int main() {
    int i;
    for (i = 33 ; i < 127 ; i++)
        g_print("%3d = [%c]\n",i,i);

    return 0;
}
```

O que este programinha fará é listar toda a tabela a partir do código 33 (“!”) até o código 126 (“~”).

Pra compilá-lo você usará uma linha similar a que estamos usando para compilar e rodar o nosso teste da glist:

```
$ gcc ascii.c -o ascii `pkg-config --cflags --libs glib-2.0`
$ ./ascii
```

Nós usaremos a função “g_list_sort()” e para isso vamos definir o protótipo da nossa função de comparação logo abaixo do protótipo da função “mostra()” (linha 12):

```
gint compara(gconstpointer a,gconstpointer b);
```

e o corpo da função será digitado lá no final do arquivo, após a função mostra (linha 38).

```
gint compara(gconstpointer a,gconstpointer b) { //Compara duas strings
    return g_ascii_strncasecmp( (const gchar *) a , (const gchar *) b , 5) ;
}
```

Observe que o protótipo da função corresponde ao da função “GcompareFunc” e a comparação é feita na realidade pela função “g_ascii_strncasecmp()” que é usada para comparar strings. Esta função é a substituta da função “g_strncase()” que tornou-se obsoleta e por isso não deve ser usada em novos códigos.

Consultando no manual, por estas funções vemos que elas recebem dois ponteiros para “const char*” e o número de caracteres que serão usados para comparação, no nosso caso serão apenas os 5 primeiros.

Agora que já temos a função definida só precisamos chamá-la junto com o “g_list_sort()” para que tenhamos a nossa lista ordenada.

Vamos incluir o código abaixo logo acima do “g_list_for_each()” que usamos anteriormente para exibir o conteúdo da lista (linha 25).

```
MinhaLista = g_list_sort(MinhaLista,compara); //Ordena a lista
g_print("\nLista Ordenada\n");
```

Com isso estamos ordenando a lista e em seguida exibindo-na na tela.

Agora compile e rode o programa para ver o resultado.

```
$ gcc main.c -o testeglist `pkg-config --cflags --libs glib-2.0`
$ ./testeglist Linux GNU GPL Debian Glib Kernel

Teste da GList
[Total de elementos: 6]

Lista Ordenada
* Debian
* Glib
* GNU
* GPL
* Kernel
* Linux
```

Observe que a lista foi corretamente ordenada e mesmo tendo indicado que a ordenação ocorre até o 5º caracter de cada elemento mesmo os itens com menor tamanho foram ordenados certos. Saiba ainda que se tivesse algum elemento com tamanho maior ele seria ordenado normalmente também.

Vale ainda uma nota a respeito da função “g_list_insert_sorted” que pode se aproveitar da nossa função “compara” para inserir os itens já ordenados na nossa lista. Assim poderíamos reduzir a quantidade de comandos no nosso programa substituindo o “append” seguido do “sort” por apenas o “insert_sorted”.

Se quiser experimentar (eu recomendo fazer isso, para fixar) então comente a linha 21 (“g_list_append”) e insira abaixo dela a linha a seguir, depois comente a linha do comando “g_list_sort” (linha 26)

```
MinhaLista = g_list_insert_sorted(MinhaLista, (gpointer) argv[i],
compara); //Insere ordenando
```

Com isso todo elemento que for inserido à lista entrará na posição adequada.

Invertendo a lista

Para invertermos a lista existe uma maneira bem simples que é usando a função “g_list_reverse()” como mostrado abaixo.

```
MinhaLista = g_list_reverse(MinhaLista); //Inverte a lista
```

Inclua isso após a função de ordenação (linha 26) e veja o resultado. Mais simples que isso é pedir demais! É claro que pra esse caso não haveria essa necessidade, pois se queremos apenas classificar em ordem decendente (de Z para A) poderíamos incluir um sinal de menos “-” no comando “return” da função “compara” como abaixo:

```
return -g_ascii_strncasecmp( (const gchar *) a , (const gchar *) b , 5) ;
```

A função “g_list_reverse” é dedicada aos casos onde você precisa inverter o conteúdo da lista independente da ordem de classificação. Por exemplo a listagem vinda de um arquivo com histórico de endereços num navegador web, onde você deseja exibir do último (acessado recentemente) para o primeiro (mais antigo).

Removendo um elemento da lista

Para remover elementos da lista podemos usar as funções “g_list_remove()” ou “g_list_remove_all()”. A diferença entre elas é que havendo dois ou mais elementos com o mesmo conteúdo a primeira

função removerá apenas o primeiro elemento da lista e a segunda função removerá todos.

Por exemplo, suponha uma lista contendo: {A B C D E F A B C D E A B C D}

Se usarmos a função “g_list_remove” para remover o elemento “A” a lista passará a ser: {B C D E F A B C D E A B C D}, enquanto que se usarmos a “g_list_remove_all” a lista passará a ser: {B C D E F B C D E B C D}.

O protótipo das funções é o que segue abaixo:

```
GList* g_list_remove (GList *list, gconstpointer data);
```

e

```
GList* g_list_remove_all (GList *list, gconstpointer data);
```

Ambas precisam receber a lista a ser alterada e o conteúdo do elemento a ser removido. Veja um exemplo simples abaixo:

```
MinhaLista = g_list_remove(MinhaLista, (gconstpointer)argv[1]);
```

Neste exemplo estamos removendo o ultimo elemento inserido na lista.

Mas vale destacar aqui um problema que encontramos com frequencia. Insira esta linha logo abaixo do “} //for i” (linha 24)

Compile e rode o programa. Observe que o ultimo elemento inserido (“Linux”) foi removido. Certo?!

```
$ gcc main.c -o testeglist `pkg-config --cflags --libs glib-2.0`
$ ./testeglist Linux GNU GPL Debian Glib Kernel

Teste da GList
[Total de elementos: 5]

Lista Ordenada
* Debian
* Glib
* GNU
* GPL
* Kernel
```

Agora substitua a variável “argv[1]” pelo próprio valor, ou seja “Linux” como abaixo:

```
MinhaLista = g_list_remove(MinhaLista, (gconstpointer)“Linux”);
```

Compile e rode. Veja que a lista ficou intacta, ou seja o “g_list_remove” por algum motivo que parece inexplicável não removeu o elemento.

Por que isso aconteceu?! Simples! Se você olhar como nós inserimos os valores na lista, verá que o seu elemento “data” não contém as strings em si, mas apenas um ponteiro indicando onde está o “argv[]” correspondente. Logo, não adianta indicar o texto, mas sim o endereço.

Uma forma de resolver esse problema é usar o foreach para percorrer a lista e em cada elemento ele pega o item e compara o valor desejado. Apesar de estranho essa forma é a mais comum de se usar, pois você geralmente usará uma estrutura como por exemplo uma “struct” com vários dados, ao invés de uma simples string.

Vejam a solução:

Acrescente esse protótipo junto dos demais:

```
void remover(gpointer item, gpointer valor); //Remove item
```

No final do arquivo inclua o corpo da função:

```
void remover(gpointer item, gpointer valor) { //Remove item
    if (compara(item, valor) == 0)
        MinhaLista = g_list_remove(MinhaLista, item);
}
```

Agora seja feliz! Inclua uma chamada para o foreach no lugar daquele `g_list_remove` que não funcionou (linha 25).

```
g_list_foreach(MinhaLista, remover, "Linux");
```

Veja que nós aproveitamos a função “compara” que já havíamos criado. Isso nos garante que se mudarmos o tipo de dado principal da nossa lista a remoção estará atualizada também. O seu funcionamento depende do “foreach” que chama a nossa função “remove” passando a ela o campo “data” do elemento atual e o valor que queremos excluir. Ao fazer isso a função compara se ambos são idênticos (retorna 0 se forem iguais). Se realmente for o mesmo então o item é removido.

Você poderá experimentar agora que independente de passar uma constante como “Kernel”, “Linux” etc. ou uma variável como “argv[1]” o item será localizado e removido.

Esta construção só tem um problema: Como o “foreach” é executado em todos os elementos, independentemente se você usar a função “remove” ou a função “remove_all” todos os elementos com aquele texto serão removidos. Para resolver esse problema, se isso for inconveniente para a sua aplicação é possível fazermos um loop manualmente com um laço while e após encontramos o item a ser removido quebramos esse laço com o “break” do “C/C++”.

Compile e rode o programa passando o primeiro argumento repetido várias vezes:

```
$ gcc main.c -o testeglist `pkg-config --cflags --libs glib-2.0`
$ ./testeglist Linux GNU GPL Debian Glib Kernel Linux Linux
Teste da Glist
[Total de elementos: 5]

Lista Ordenada
* Debian
* Glib
* GNU
* GPL
* Kernel
```

Observe que o argumento “Linux” que foi passado três vezes foi completamente extinto da lista. Veja a solução para este problema.

Nós vamos criar uma função seguindo o que foi explanado acima. O protótipo abaixo mostra que a nossa função é bem similar ao “`g_list_remove`” da Glib, mudando apenas o valor que na função original é um “constpointer” e esta nossa função eu usei o tipo mais adequado ao programa em questão que é um “const gchar*”, além do retorno que o normal era ponteiro para a nova lista e nessa função personalizada ela retorna “void”. Você poderá incluí-lo junto aos demais protótipos de funções que já temos.

```
void g_list_remove_em(GList *lista , const gchar *valor); //Remove um item
```

O corpo da função é este e pode ser incluso no final do nosso arquivo:

```
void g_list_remove_em(GList *lista , const gchar *valor){ //Remove um item
    lista = g_list_first(lista);
    do
        if (compara(lista->data, valor) == 0) {
            lista = g_list_remove(lista, lista->data);
            break;
        }
    while (lista = g_list_next(lista));
}
```

E a forma de chamada é esta e você poderá incluí-lo no lugar daquele foreach que usamos para

remover elementos, você poderá experimentar substituir o “argv[1]” por um elemento constante qualquer e ver o resultado:

```
g_list_remove_em(MinhaLista, "Linux");
```

Vejam que não é uma função complexa. Na chamada nós passamos a lista e o texto a ser removido. A primeira coisa que a função faz é garantir que estamos no início da lista (função “g_list_first”) e fazer um loop enquanto há um próximo elemento (função “g_list_next”). Durante a iteração deste laço nós verificamos se o componente “data” da nossa lista é o mesmo do valor passado. Se for nós o removemos com o “g_list_remove” e quebramos (break) o laço de forma a garantir que mesmo existindo outro elemento com esse mesmo texto ele não será removido.

Compilando e rodando o programa temos o seguinte resultado:

```
$ gcc main.c -o testeglist `pkg-config --cflags --libs glib-2.0`
$ ./testeglist Linux GNU GPL Debian Glib Kernel Linux Linux
Teste da GList
[Total de elementos: 7]

Lista Ordenada
* Debian
* Glib
* GNU
* GPL
* Kernel
* Linux
* Linux
```

Observe agora que mesmo tendo passado o parâmetro “Linux” três vezes apenas o primeiro foi removido.

Conclusão

Como vocês puderam ver trabalhar com listas encadeadas pode ser um processo bem complexo usando-se apenas os recursos de ponteiros do C/C++, mas por outro lado pode ser bem prático usando o tipo GList da Glib e as funções já criada especificamente para isso.

Dê uma olhada na seção de exemplos de uso dos Widgets que assim poderão ver mais alguns exemplos práticos de uso da GList com os Widgets do GTK+.

Encerramento

Este documento está em constante atualização portanto é sempre bom que o leitor esteja atento as novas alterações e inclusões que surgirão. Para facilitar a identificação de uma versão mais antiga para outra mais recente será usada a notação:

“nome-do-arquivo-ano-mes-dia.tipo”

Ex.:

tutorialglade-20040414.pdf

Marcas registradas

Windows é marca registrada da Microsoft Corp (<http://www.microsoft.com>), Borland, Delphi, Kylix e C++Builder são marcas registradas da Borland/Inprise, GNU é marca registrada da GNU (<http://www.gnu.org>). Linux é marca registrada de Linus Torvald (<http://www.linux.org>) outras

marcas citadas pertencem aos seus respectivos donos.

Licença de Documentação Livre GNU

Nota: Uma tradução, não oficial, desta licença para o “português Brasil” pode ser lida em <http://www.ead.unicamp.br/minicurso/bw/texto/fdl.pt.html>

GNU Free Documentation License

Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as

Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you

must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- **B.** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- **C.** State on the Title page the name of the publisher of the Modified Version, as the publisher.
- **D.** Preserve all the copyright notices of the Document.
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- **G.** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- **J.** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- **K.** For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- **L.** Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- **M.** Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- **N.** Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- **O.** Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a

version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.