Sumário

Créditos

Anjuta um breve tutorial

Este texto não visa ser um substituto para o manual ou tutorial oficial que encontra-se disponível no menu ajuda e nem tão pouco ser uma tradução ou transliteração dos mesmos, apesar de se valer da tradução de alguns trechos e exemplos destes e outros documentos; mas apenas ser uma breve introdução ao uso desta poderosa ferramenta para desenvolvimento em C/C++ com GTK e Glade com base nos conhecimentos do autor deste texto (eu) e que apesar de serem poucos serão de grande valia para os iniciantes em Glade, GTK e C/C++.

Definição

Anjuta é uma IDE (Integrated Development Environment – Ambiente de desenvolvimento integrado) para desenvolvimento de aplicações em C/C++. Ele foi concebido pelo Sr. Naba Kumar e escrito inteiramente com GTK+/GNOME. Sendo distribuído sob a licença GPL é uma das ferramentas para desenvolvimento mais conhecidas dos programadores de aplicações para GTK+ e GNOME em estações com GNU/Linux.

De forma grosseira o Anjuta é uma interface gráfica para diversas ferramentas que deveriam ser usadas via linha de comando pelo programador, tais como o make, gcc, gdb, cvs entre outros. A vantagem de usá-lo ao invés daquelas ferramentas diretamente é que este último é muito mais flexível e simples do que usar todas aquelas ferramentas via linha de comando tornando assim o trabalho de desenvolvimento de um projeto muito mais produtivo e amigável.

Para download dos fontes ou maiores informações sobre o projeto você poderá visitar o site oficial do projeto que encontra-se em http://www.anjuta.org.

Nota: No momento em que este texto está sendo finalizado (junho/2005) já há uma nova versão saindo do forno (versão 2.0) e que trará muitas melhorias em relação a atual, mas como ela ainda é estável o suficiente para uso em ambiente de produção, todo este texto foi escrito com base na atual versão 1.2.3

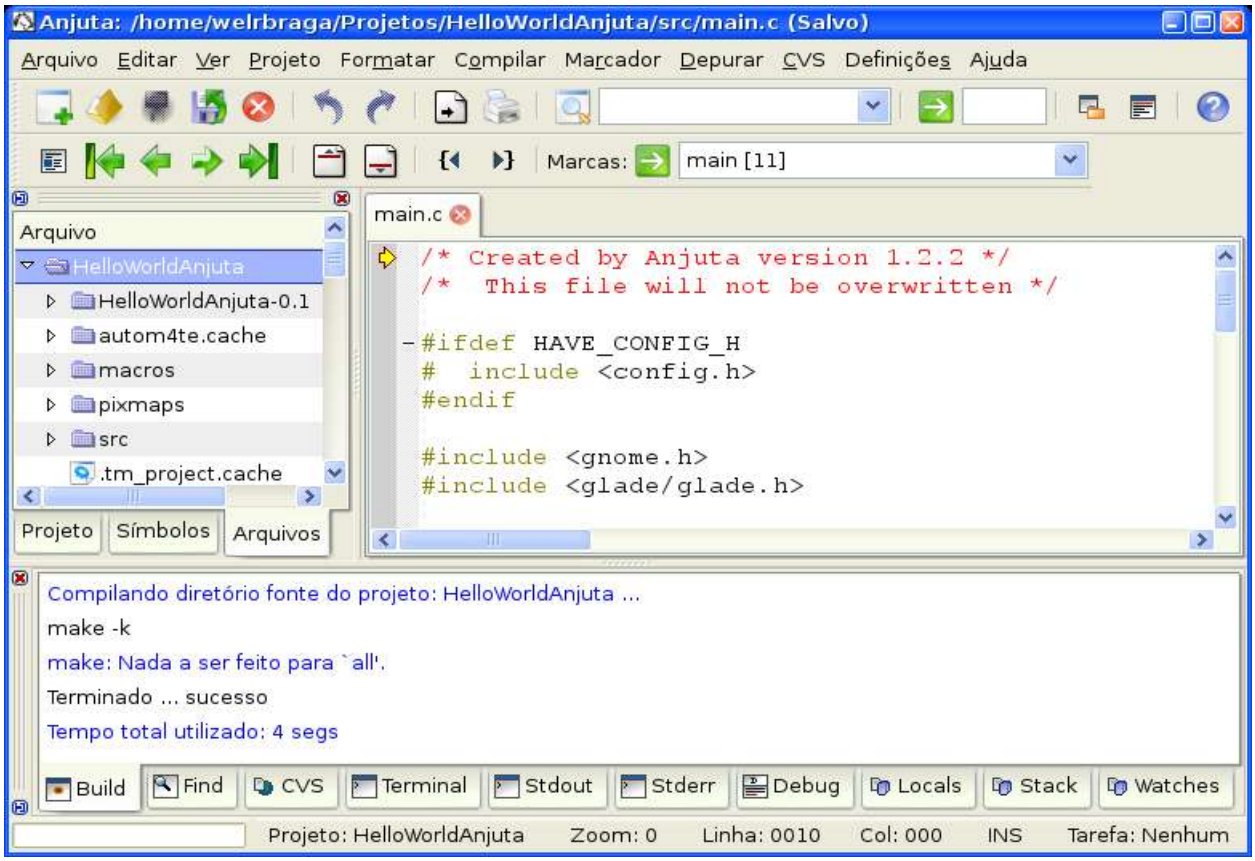

Ilustração 1 - Tela Principal do Anjuta exibindo seus principais recursos

Recursos do Anjuta

O Anjuta possui ótimos recursos que possibilitam o desenvolvimento de forma bem cômoda para o programador. Entre estes recursos eu faço questão de citar os seguintes:

- **Realce de sintaxe** – Este recurso é simplesmente fantástico! Claro que ele existe em nove a cada dez editores de texto existente no GNU/Linux, mas não custa nada frisar que este recurso também se faz presente no nosso Anjuta. Para quem não conhece este recurso, este é o que permite que as palavras reservadas de uma linguagem sejam apresentadas com cores diferentes para facilitar a leitura do código.
- **Autocompletar e Exibição de Sintaxe** – Assim como acontece no OpenOffice, Microsoft Word e até mesmo em algumas IDEs comerciais para desenvolvimento, durante a digitação do código o Anjuta apresenta uma lista de funções, variáveis etc que iniciam com as letras já digitadas e após a digitação de todo nome da função ele apresenta numa "caixinha" a sua sintaxe. Isso reduz enormemente o risco do programador obter um erro de compilação simplesmente por que esqueceu uma letra ao digitar o gigantesco no nome daquela função "*gtk_alguma_coisa_em_algum_lugar()*"!
- **Navegador de projeto** – Quem desenvolve programas organizados em vários arquivos e nunca teve esse recurso no seu editor de textos vai se sentir feliz ao saber que agora terá todos os arquivos de fontes do seu projeto ao alcance de um clique. Este não é um recurso exclusivo do Anjuta mas que, em conjunto com a abertura de múltiplos arquivos em abas, ajuda bastante e se você nunca o usou, depois que começar a usá-lo ficará se perguntando porque não usou antes.
- **Ferramentas de navegação rápida** – Estas ferramentas são fantásticas. Com elas você

poderá navegar facilmente pelo seu código escolhendo o nome da função desejada em uma caixa de combinação ou simplesmente digitar o número da linha desejada ou ainda procurar por um texto enquanto ele está sendo digitado. Apesar destes recursos funcionarem apenas no arquivo atualmente em foco, você poderá usar o navegador de projeto comentado anteriormente para navegar entre as funções de todos os arquivos no seu projeto.

- **Numeração de linhas** – Este é um recurso simples, porém indispensável para trabalhar com programação. Imagine, por exemplo, no meio da compilação aparecer aquela frustrante mensagem acusando erro de sintaxe na linha 5432. Sem a numeração de linhas você teria que contar as linhas uma a uma desde o início do arquivo até este ponto!
- **Integração com o Glade** – Aqui está um recurso que torna o Anjuta muito poderoso para desenvolver aplicações para o GTK+ e para o Gnome. Com uma simples combinação de teclas (ou usando o menu) você poderá abrir o arquivo ".glade" associado ao seu projeto dentro do Glade, alterá-lo a vontade, salvar e voltar ao seu código despreocupadamente. Vale aqui uma nota sobre isso, pois quem prefere deixar o Glade gerar o código fonte ao invés de trabalhar com a libglade terá algumas dores de cabeça, já que o código gerado pelo Glade é reescrito toda vez que você altera o arquivo ".glade" e manda gerar o código fonte. Como a maioria das aplicações para GNOME atualmente usam libglade ao invés da geração de código para interface, então você pode usá-la despreocupadamente.
- **Integração com o Devhelp e outras ferramentas de ajuda** – O programador que nunca precisou abrir abrir o manual da sua linguagem de programação ou algum outro recurso de ajuda que remova a tecla F1 do seu teclado e a atire em mim! O Devhelp é o navegador padrão para ajuda de desenvolvimento no GNOME. A partir dele o programador tem acesso a ajuda para todas as principais APIs, bibliotecas e programas que sejam necessários e se por acaso faltar alguma basta abrir a página do projeto Lidn (http://lidn.sourceforge.net) e baixar o documento apropriado para integrar ao Devhelp.
- **Terminal embutido** – Este é um recurso que apesar de não ser tão essencial traz suas vantagens. É possível ter um terminal aberto a todo instante dentro de uma aba no seu Anjuta para emissão de comandos que por ventura precisem ser executados rapidamente.
- **Construtor de classes –** Se você gosta de programar em C++ com recursos de orientação a objetos então o Anjuta pode te ajudar a construir as classes do seu projeto com este assistente que apesar de ser bastante rudimentar já facilita um pouco o seu trabalho, principalmente na questão de dividir cada classe em um arquivo separado.

Estes recurso apresentados acima foram apenas alguns dos muitos que esta poderosa ferramenta possui. Seria muito entediante, tanto pra mim quanto pra você, continuar escrevendo/lendo sobre todos eles, por isso os demais recursos serão comentados ao longo deste tutorial, quando houver necessidade.

Antes de começar

Daqui pra adiante eu vou considerar que o Anjuta já esteja instalado e todas as bibliotecas necessárias estejam instaladas para desenvolvimento de aplicações com o GTK+, Glade e GNOME. Se você não tiver estes pacotes instalados deverá instalá-los e se não souber como instalar, recomendo parar a leitura aqui e procurar ajuda em sites e fóruns apropriados para saber como fazer isso.

Antes que alguém critique o fato de eu não explicar o procedimento de instalação eu vou me justificar: Cada distribuição GNU/Linux possui um modo diferente de instalação de pacotes e resolução de dependências. Não seria muito prático escrever cada um deles e nem justo com todas as excelentes distros existentes por ai se explicasse o método para uma ou outra. Eu particularmente uso o GNU/Debian, onde posso instalar todos os pacotes de dependências

usando a ferramenta Synaptic (http://www.nongnu.org/synaptic) que também está disponível em distros como o Kurumin, Kalango, Knoppix, Ubuntu, Conectiva entre outras. Caso precise de ajuda você poderá recorrer até mesmo ao fórum do nosso site (http://www.gtk-br.cjb.net).

Vale aqui destacar também que na maioria das vezes a instalação do Anjuta é simples e por isso bem sucedida, mas o usuário terá problemas para compilar projetos que necessitem de alguma biblioteca que não foi instalada por não ser considerada como uma dependência. Como sugestão para evitar transtornos, se você tiver um bom espaço disponível em disco instale todos os pacotes que contenham a terminação "dev" (ou devel, em algumas distros), por exemplo se você pretende desenvolver aplicações que usem o Mesa/OpenGL precisará do pacote "libgtkgl" para execução das aplicações e do pacote "libtkgl-dev" para compilar as aplicações. Talvez a sua distro use uma nomenclatura diferente para especificar os pacotes e desenvolvimento, informe-se com alguém que já tenha experiência com ela.

Uma outra forma de começar seria instalando os *meta-pacotes*[1] de desenvolvimento para gnome da sua distro, no caso do Debian, com Gnome 2.8, isso corresponde aos pacotes gnome-devel e gnome-core-devel, algumas outras distros usam o conceito de tarefas cuja a finalidade é a mesma.

O primeiro projeto

A título de primeiro projeto nós iremos criar um tradicional *hello world* que apenas apresenta uma Janela do Gnome na nossa tela. Depois nós iremos brincar com alguns dos recursos dos Anjuta sem nos prendermos aos recursos da linguagem C e das bibliotecas que forem usadas, já que este não é o objetivo deste documento.

A primeira coisa a fazer é iniciar o Anjuta. Se você usa o GNOME ele encontra-se na pasta "**Desenvolvimento**" do menu "**Aplicações**", mas ele pode ser chamado digitando-se o comando "**anjuta**" a partir de um terminal ou a caixa de entrada do comando "**Executar Aplicação**".

Assim que o Anjuta estiver rodando você verá uma janela como a da Ilustração 3

1 Meta-pacote é um tipo de pacote especial que ao invés de possuir os arquivos para instalar algum programa, biblioteca ou documentação ele possui apenas indicação de dependências de maneira que ao ser instalado ele "força" a instalação de todos os pacotes relacionados a ele. Por exemplo o meta-pacote "gnome" não possui todo o sistema Gnome, mas apenas as informações de dependências que apontam para os pacotes reais. Algumas distros ao invés deste conceito usam o conceito de "tarefas" onde ao instalar uma tarefa todos os pacotes relacionados a ela serão instalados.

Ilustração 2 - Caixa de diálogo inicial do Anjuta

Neste diálogo você poderá escolher uma dentre as opções seguintes:

- **Assistente de Aplicação** – É por aqui que você começa uma aplicação de forma bem fácil e agradável. Usando o assistente de aplicação você poderá escolher o tipo da aplicação, definir algumas propriedades e depois começar a programar. Esta opção equivale ao comando **Novo Projeto** do menu **Arquivo**, portanto se você deseja usar esta opção mas esta caixa de diálogo não foi exibida então você poderá usar este comando.
- **Assistente de importação** – Este é um recurso interessante que apesar de não ser 100% eficaz, quando funciona, facilita bastante o trabalho de importar um projeto feito em outro ambiente para o Anjuta. Ele também é útil para transformar aqueles seus projetos que você gerencia braçalmente em projetos gerenciáveis pelo Anjuta. Esta opção também está disponível pelo comando **Importar Projeto** do menu **Arquivo**.
- **Abrir Projeto** – Com esta opção você poderá abrir um dos seus projetos salvos para continuar o trabalho. A ação equivalente a esta é a executada pelo comando **Abrir Projeto** do menu **Arquivo**.
- **Abrir último projeto utilizado** – O nome já diz tudo! Pra que escolher a opção "Abrir Projeto" e ter que escolher o último projeto se você pode fazer isso com um único clique?! Esta opção também está acessível pelo menu **Arquivo** onde existe uma lista com os últimos projetos sob a opção **Projetos Recentes**.
- **Abrir Arquivo** – É muito comum criarmos algum "programinha" simples em um único arquivo só para resolver algum problema e não se quer gerar um projeto no Anjuta que depende de algumas dezenas de arquivos e bibliotecas. Neste caso você poderá simplesmente

abrir o seu arquivo ".c" e editá-lo com todo o conforto que o Anjuta oferece. Você poderá abrir também algum arquivo de anotações ou com dados que serão usados pela sua aplicação. Isso equivale ao comando **Abrir** do menu **Arquivo**.

- **Novo Arquivo** – Seguindo a mesma idéia do exemplo citado para opção abrir Arquivo, você poderá usar esta opção para criar um pequeno arquivo de programa ou dados para sua aplicação. Isso equivale ao comando **Novo** do menu **Arquivo**.

Nós escolheremos a opção **Assistente de Aplicação** para começarmos um novo projeto. Este assistente é bem simples e possui apenas alguns passos essenciais para gerar a nossa aplicação. Todas as demais opções do projeto deverão (se necessário) ser alteradas a partir do comando **Configurar Projeto** do menu **Projeto**.

As telas a seguir são do nosso assistente. Acompanhe passo-a-passo como criar um novo projeto:

A primeira tela (Ilustração 2) é apenas uma página introdutória com uma breve explicação e a única coisa a ser feita aqui é clicar no botão **Avançar**.

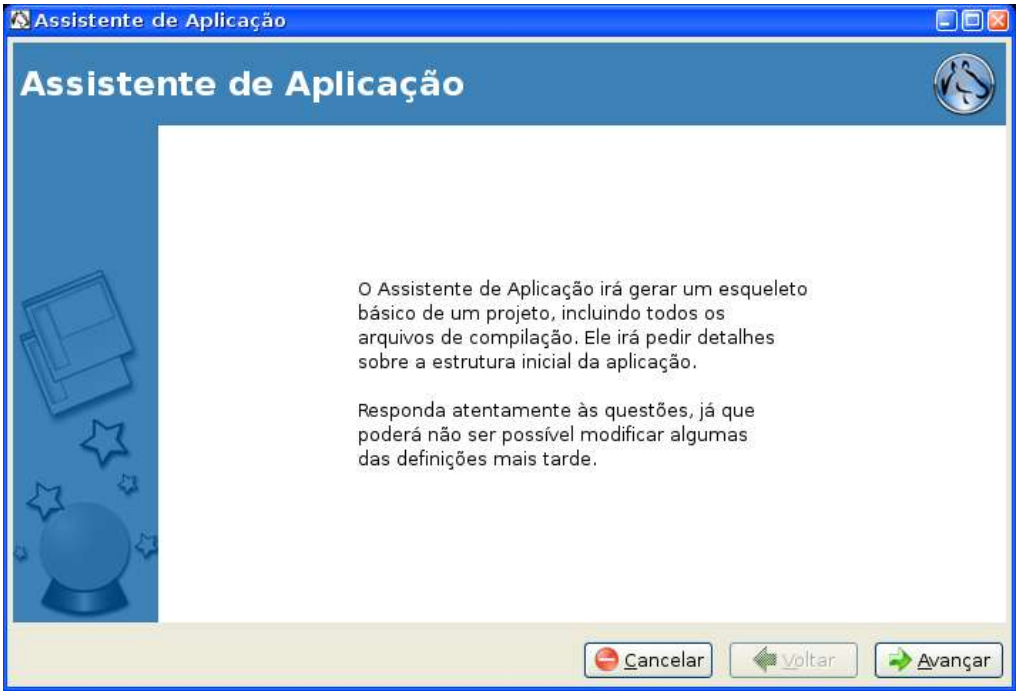

Ilustração 3 - Assistente de Aplicação (página 1)

A próxima tela (Ilustração 4) permite escolher o tipo de projeto a ser criado. Para efeitos práticos no nosso tutorial vamos usar a opção *Projeto libGlade 2.0* que nos permite criar aplicações gráficas para o Gnome usando esta biblioteca. Lembre-se que para que esta opção seja usada com sucesso todas as bibliotecas de desenvolvimento para o Glade, libGlade e Gnome devem estar instaladas no seu sistema, este aviso é válido para todas as outras opções em que os pacotes de desenvolvimento relacionados devem estar instalados. Nesta tela nós simplesmente escolhemos a opção desejada e clicamos em **Avançar**.

Ilustração 4 - Assistente de Aplicação (página 2)

A terceira tela (Ilustração 5) é destinada a informar dados do programa, tais como o nome do projeto (que não deve possuir espaços), o número de versão (o padrão é começar com 0.1), o nome do autor (aqui você põe o seu nome), alvo do projeto (o nome do arquivo executável definido automaticamente de acordo com o nome do projeto). A linguagem de programação pode ser "C", "C++" ou ambos, como nós escolhemos o projeto para a libGlade aqui nós devemos escolher a linguagem C, já que para programarmos em C++ com a libGlade seria necessária a biblioteca libglademm e que o Anjuta ainda não suporta completamente. O tipo de alvo poderá ser um alvo executável (programa comum), ou uma biblioteca (estática ou dinâmica). Preencha esta tela conforme desejado e clique em **Avançar**.

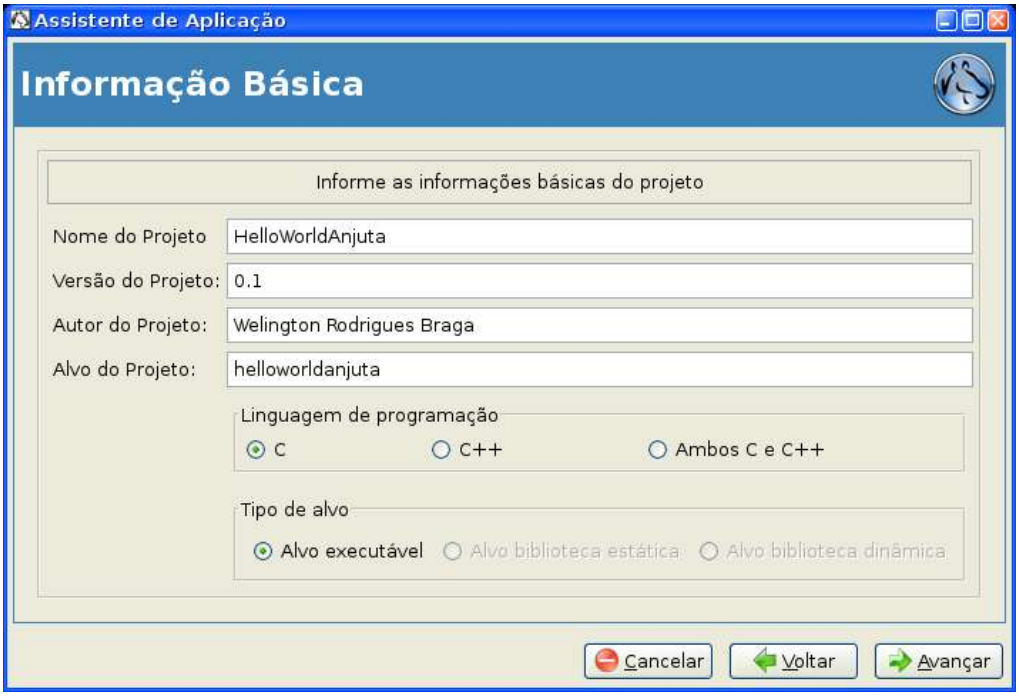

Ilustração 5 - Assistente de Aplicação (página 3)

A quarta tela do assistente (Ilustração 6) permite que você escreva um texto descrevendo o seu projeto. Aqui você pode escrever a vontade e quando terminar clique em **Avançar**.

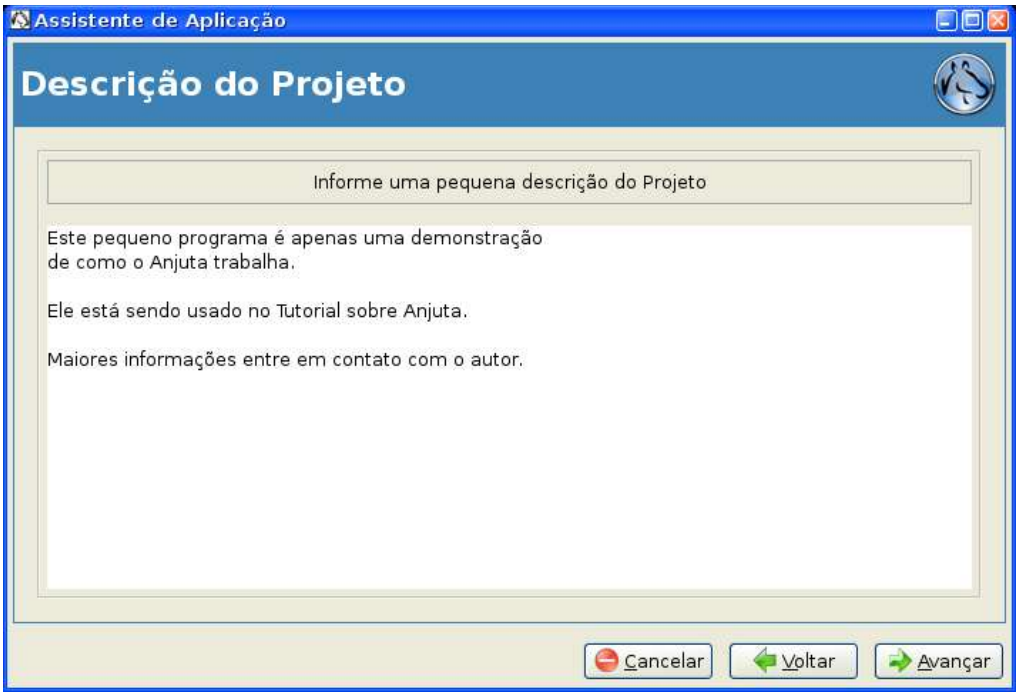

Ilustração 6 - Assistente de Aplicação (página 4)

A próxima tela (Ilustração 7) é para passar algumas opções adicionais à sua nova aplicação tais como incluir uma cópia resumida da licença GPL em todos os arquivos (isso é recomendável se você pretende criar uma aplicação sob esta licença), Habilitar o suporte a gettext é indispensável se você pretende fazer uma aplicação "já prevendo o sucesso" e que futuramente será internacionalizada. A opção de gerar o código fonte usando o Glade ou glademm está disponível apenas para projetos baseados diretamente na GTK o que permitirá o Anjuta aproveitar o código fonte gerado pelo Glade.

As opções Nome da Entrada, Comentário, Grupo, Usar terminal e Ícone são usadas para criar o arquivo "`seu_projeto.desktop`" que é um atalho para o seu programa e que será criado dentro do grupo especificado, quando o usuário der um "make install" na sua aplicação.

Defina as opções como desejado e clique em **Avançar**.

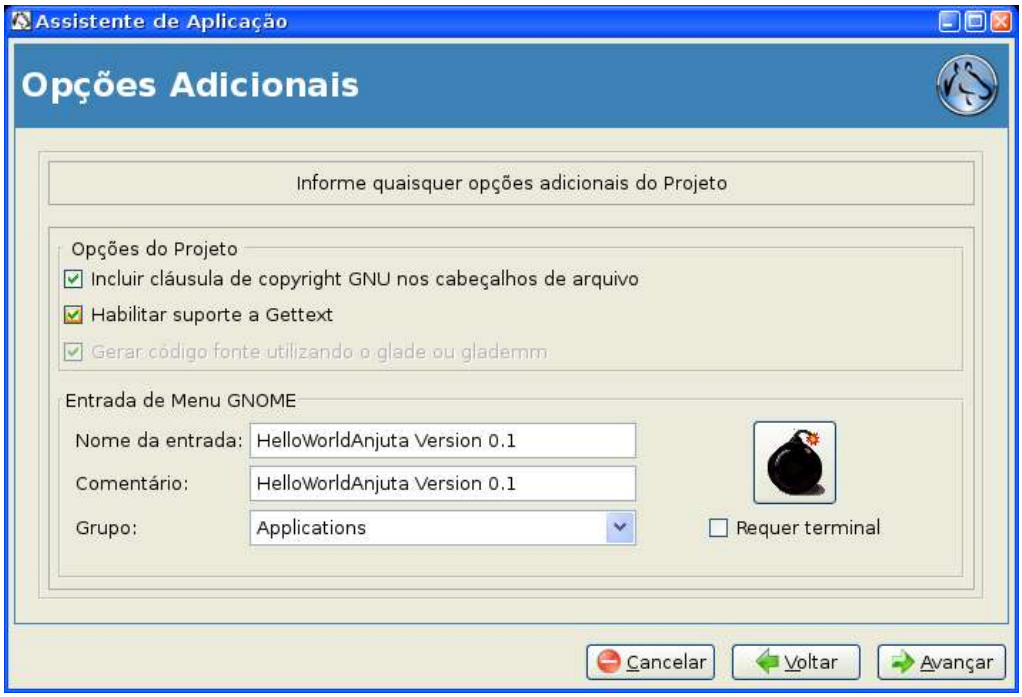

Ilustração 7 - Assistente de Aplicação (página 5)

A próxima tela (Ilustração 8) apresenta um resumo das opções que você escolheu. Se estiver tudo conforme você desejava que estivesse então clique no botão **Aplicar** para que o Anjuta finalize o projeto e gere todos os arquivos necessários.

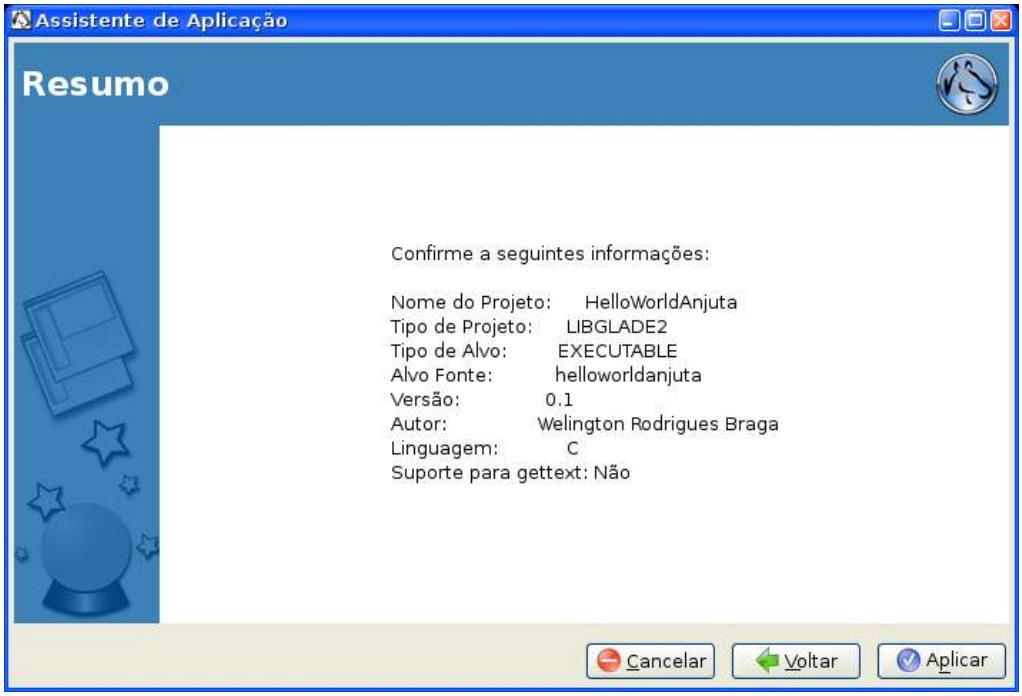

Ilustração 8 - Assistente de Aplicação (página 6)

Se tudo correu bem as últimas linhas que se pode ver na ficha “Build” da janela de mensagens (veja a na página) deve ser algo similar ao mostrado abaixo:

```
Agora digite 'make' para compilar o pacote
Terminado ... Sucesso
```

```
Tempo total utilizado: 45 segs
```

Estas mensagens foram geradas pelos programas automake e autoconf que são responsáveis por criar os arquivos makefile e outros arquivos auxiliares para facilitar a compilação. Dependendo da versão destes pacotes, que estiverem instalados em sua máquina, a mensagem será um pouco diferente.

Uma vez que já temos a base do projeto pronta, nós podemos compilar o nosso projeto ... ou a partir do terminal (você pode clicar na ficha terminal da janela de mensagens), ou usando o comando **Compilar Tudo** do menu **Compilar** [SHIFT-F11].

Ao concluir a compilação nós teremos um protótipo do nosso projeto já funcional e para vê-lo basta usar o comando **Executar** do menu **Compilar** [F3].

Se tudo realmente correu bem você terá a janela da sua aplicação e uma janela de terminal, conforme a ilustração a seguir:

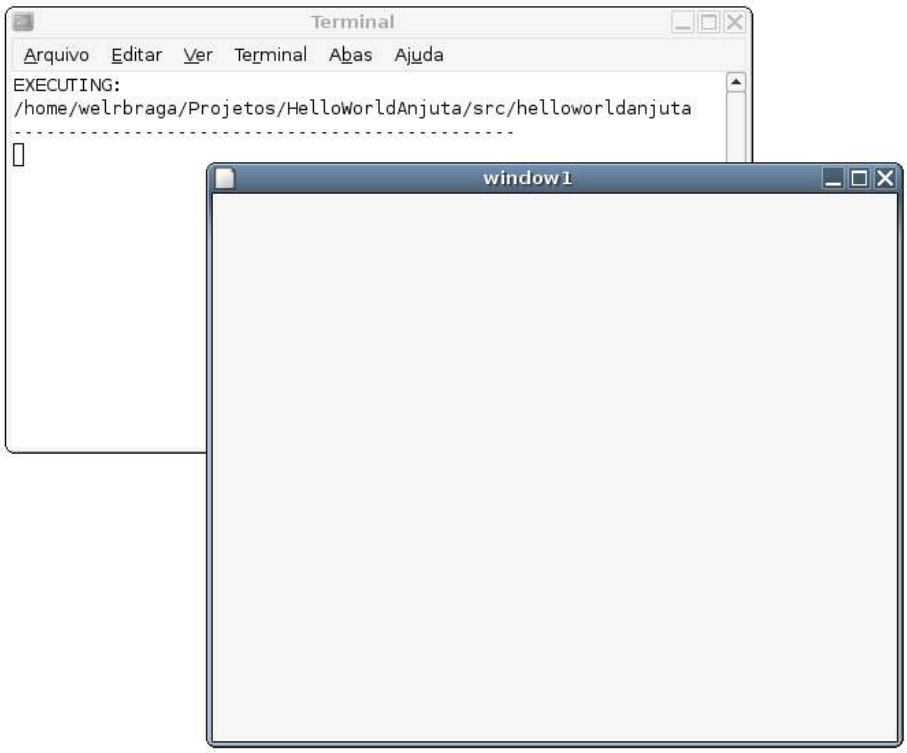

Ilustração 9 - Primeira execução do nosso Hello Anjuta

NOTA: Sempre que iniciarmos a nossa aplicação de dentro do Anjuta, isto é, usando o comando **Executar** do menu **Compilar** ou a tecla [F3] a janela do terminal será aberta atrás de nossa aplicação para que possamos ver as mensagens de depuração; mas fique despreocupado pois se você executar a aplicação de fora da IDE este terminal não será aberto.

Para fecharmos a nossa aplicação você deve fechar a janela de terminal (lembre-se que a janela da aplicação ainda não possui o evento `gtk_quit()` associado ao clique do botão fechar).

O básico é isso e certamente que a maioria das pessoas que leram ou estão lendo este documento sabem chegar até aqui, mas não sabe como prosseguir e é por isso que eu pretendo me alongar detalhando alguns dos vários recursos do Anjuta.

Incrementando o projeto

Até aqui o nosso projeto está bastante genérico e qualquer programa usando a libGlade, no Anjuta começará desta forma. Daqui pra frente depende do que o programador espera desenvolver. No nosso caso faremos uma desprezível calculadora com apenas as quatro operações básicas. A minha proposta não é produzir nenhuma super-calculadora e nem mesmo igual ou comparável as calculadoras do Gnome, KDE ou Windows®. Mas apenas um aplicativo onde possamos testar o potencial do nosso Anjuta.

Seu funcionamento será o seguinte: Teremos duas caixas de entrada, onde digitaremos os valores usados nos cálculos, ao clicarmos em um dos botões de operação teremos o resultado exibido em um rótulo. Apenas isso e nada mais. Vamos lá então!

Invocando os poderes do Glade

Como já dito antes o Anjuta é usado para desenvolvimento do código fonte e gerenciamento dos arquivos, mas para desenvolvimento da interface gráfica nós dependemos do Glade, por isso certifique-se de que o Glade esteja instalado e funcionando no seu sistema.

Uma vez que o Glade esteja instalado basta teclar [**ALT**]+[**G**] ou no menu **Projeto** escolher o comando **Editar GUI da Aplicação**. Com isso o Glade será aberto e nós poderemos incluir os componentes da nossa aplicação.

NOTA: Caso você não saiba usar o Glade então recomendo fortemente que você dê uma lida no meu tutorial sobre o uso do Glade publicado no site http://gtk-br.cjb.net pois ele te dará as condições necessárias para que se desenvolva aplicações usando aquela poderosa ferramenta e que será usada aqui.

Eu vou considerar daqui pra frente que você, prezado leitor, já saiba usar o Glade e vou acelerar as coisas nesta etapa, pois não é o nosso objetivo perder tempo com aquela ferramenta que já foi explanada em outro documento.

A nossa aplicação deverá ficar com a aparência mostrada na Ilustração 9

Ilustração 10 - Screenshot da calculadora completa

Observe que ela é bem simples e por isso dispensa muitos comentários sobre a sua elaboração, mas para ajudar aos "marinheiros de primeira viagem" ai vai algumas dicas:

1. Use *GtkVBox* (Caixa Vertical) com 4 linhas para organizar melhor os componentes.
2. Inclua duas *GtkEntry* (Entrada de Texto), uma para cada valor da nossa operação
3. Ao invés de quatro botões use apenas uma *GtkHButtonBox* (Caixa de botões Horizontal) com 4 botões.
4. Insira uma *GtkLabel* (etiqueta, ou rótulo)

As únicas propriedades alteradas aqui foram as propriedades "etiqueta" de cada um botões para que representassem as quatro operações básicas.

Além das propriedades foram definidos os seguintes sinais e manipuladores:

Objeto	*Sinal*	*Manipulador*
Window1	destroy	gtk_main_quit()
Button1	clicked	on_button1_clicked()
Button2	clicked	on_button2_clicked()
Button3	clicked	on_button3_clicked()
Button4	clicked	on_button4_clicked()

Tabela 1 - Sinais e Manipuladores

Resumindo a tabela: Ao clicarmos no botão fechar da janela (Window1) o sinal *destroy* é ocorrido o que chama imediatamente a função *gtk_main_quit()* como manipulador e que encerra a nossa aplicação. E ao clicarmos em qualquer um dos outros botões o sinal "*clicked*" ocorre chamando cada um dos seus respectivos manipuladores "on_buttonN_clicked()".

Com exceção do manipulador gtk_main_quit(), que é uma função interna do GTK, os demais manipuladores deverão ser programados em funções com estes nomes a partir do Anjuta (adiante nós veremos como) .

Por hora apenas salve as alterações no Glade, clicando em "Salvar" e feche-o.

Para ver como o projeto está basta executar a aplicação usando o comando **Executar** do menu **Compilar** ou a tecla [F3], como já dito antes. Lembrando apenas que o terminal surgirá por trás para que possamos depurar nossa aplicação.

Se tudo correr bem a nossa aplicação deverá aparecer na tela (lembre-se que a janela é criada em tempo de execução e por isso alterações feitas pelo Glade não necessitam de uma recompilação do projeto), mas é claro que o único botão que funciona é o "X" para fechar a janela, já que os demais deverão ser programados.

Agora observe a saída no terminal de depuração:

```
EXECUTING:
/home/welrbraga/Projetos/HelloWorldAnjuta/src/helloworldanjuta
-----------------------------------------------

(helloworldanjuta:3584): libglade-WARNING **: could not find
signal handler 'on_button2_clicked'.

(helloworldanjuta:3584): libglade-WARNING **: could not find
signal handler 'on_button1_clicked'.

(helloworldanjuta:3584): libglade-WARNING **: could not find
signal handler 'on_button4_clicked'.

(helloworldanjuta:3584): libglade-WARNING **: could not find
signal handler 'on_button3_clicked'.
```

Veja que no nosso terminal temos quatro avisos da libGlade acusando a ausência dos manipuladores 'on_buttonX_clicked'. Acredito que a partir daqui você, que ainda estava em dúvida, já pode ver uma utilidade para o terminal que se abre durante a execução do projeto no ambiente do Anjuta. Sempre que tivermos mensagens de erro ou aviso a serem exibidas, em tempo de execução, elas serão apresentadas no terminal. Vale deixar aqui uma dica que se você

precisar que o seu programa emita mensagens de depuração pode-se fazer isso com as funções `g_message(), g_debug, g_critical, g_error()` e `g_warning()`; consulte a documentação destas funções, na biblioteca glib, pois elas podem ajudar enormemente a depurar seus programas sem que você tenha o trabalho de construir uma "janelinha" só pra apresentar mensagens de erros durante a fase de testes, pra saber o que está acontecendo.

O esqueleto básico do programa

Vamos, a partir daqui, começar a "dar vida" a nossa calculadora. Com o projeto aberto, se o arquivo "main.c" ainda não estiver visível na área de trabalho do Anjuta então procure por ele no ***navegador de projeto*** e dê um duplo clique sobre o nome do arquivo, para ele seja aberto. Logo acima da função "main()" nós vamos incluir os seguintes protótipos:

```
GladeXML *xml; /* Estrutura global para que todas as funções possam
usa-lo */
/* Protótipos das funções                        */
gdouble get_valor (gchar * meu_widget);  /* Lê o valor do widget */
void set_valor (gchar * meu_widget, gdouble valor); /* Define o valor
do widget */

/* Protótipos das funções manipuladoras      */
void on_button1_clicked ();     /* Soma          */
void on_button2_clicked ();     /* Subtração     */
void on_button3_clicked ();     /* Multiplicação */
void on_button4_clicked ();     /* Divisão       */
```

Quadro 1 - Protótipos

Observe que na primeira linha incluída está a declaração para nossa estrutura XML, o motivo disso é que nós vamos usar com certa freqüência o conteúdo dessa variável para encontrarmos os objetos a serem manipulados, então para que tudo funcione direitinho comente a declaração dentro da função main().

Feito isso agora nós vamos colocar o corpo das funções no final do nosso arquivo.

```
gdouble get_valor (gchar * meu_widget)     /* Lê o valor do widget */
{
     /* Pega o widget especificado */
     GtkWidget *wid = glade_xml_get_widget (xml, meu_widget);

     /* Pega o tipo do widget */
     gchar *wid_tipo = g_strdup (G_OBJECT_TYPE_NAME (wid));

     /* Retorno do valor nos objetos  */
     const gchar *retorno = NULL;

     /* Verifica se o widget é uma caixa de entrada */
     if (!strncmp (wid_tipo, "GtkEntry", 8))
     {
          retorno = gtk_entry_get_text (GTK_ENTRY (wid));
     }
     else
```

```
	/* Verifica se o widget é um rótulo */
	if (!strncmp (wid_tipo, "GtkLabel", 8))
	{
		retorno = gtk_label_get_text (GTK_LABEL (wid));
	}

	/* Tranforma a string de retorno em um número */
	gdouble saida = g_ascii_strtod (retorno, NULL);

	g_free(wid_tipo);
	return saida;
}

void set_valor (gchar * meu_widget, gdouble valor) /* Define o valor
do widget */
{
	/* Pega o widget especificado */
	GtkWidget *wid = glade_xml_get_widget (xml, meu_widget);

	/* Pega o tipo do widget */
	gchar *wid_tipo = g_strdup (G_OBJECT_TYPE_NAME (wid));

	/* Converte o número de entrada em um string de 12 posições */
	gchar buffer[12];
	const gchar *valor_txt = g_ascii_dtostr (buffer, 12, valor);

	/* Verifica se o widget é uma caixa de entrada */
	if (!strncmp (wid_tipo, "GtkEntry", 8))
	{
		gtk_entry_set_text (GTK_ENTRY (wid), valor_txt);
	}
	else
		/* Verifica se o widget é um rótulo */
	if (!strncmp (wid_tipo, "GtkLabel", 8))
	{
		gtk_label_set_text (GTK_LABEL (wid), valor_txt);
	}
	g_free(wid_tipo);
}

void on_button1_clicked ()	/* Soma		*/
{
	g_print("Soma\n");
	gdouble e1 = get_valor ("entry1");	/* Lê o texto da caixa 1 e
retorna em número */
	gdouble e2 = get_valor ("entry2");
	gdouble r = e1 + e2;  /* Realiza o calculo em Si */
	set_valor ("label1", r);	/* Mostra o resultado */
}

void on_button2_clicked ()	/* Subtração	*/
{
	g_print("Subtração\n");
	gdouble e1 = get_valor ("entry1");	/* Lê o texto da caixa 1 e
```

```
retorna em número */
     gdouble e2 = get_valor ("entry2");
     gdouble r = e1 - e2;  /* Realiza o calculo em Si */
     set_valor ("label1", r);   /* Mostra o resultado */
}

void on_button3_clicked ()       /* Multiplicação */
{
     g_print("Multiplicação\n");
     gdouble e1 = get_valor ("entry1");    /* Lê o texto da caixa 1 e
retorna em número */
     gdouble e2 = get_valor ("entry2");
     gdouble r = e1 * e2;  /* Realiza o calculo em Si */
     set_valor ("label1", r);   /* Mostra o resultado */
}

void on_button4_clicked ()       /* Divisão       */
{
     g_print("Divisão\n");
     gdouble e1 = get_valor ("entry1");    /* Lê o texto da caixa 1 e
retorna em número */
     gdouble e2 = get_valor ("entry2");
     gdouble r = e1 / e2;  /* Realiza o calculo em Si */
     set_valor ("label1", r);   /* Mostra o resultado */
}
```

Quadro 2 – Funções adicionais

Se você digitar tudo direitinho, ou simplesmente copiar e colar, então tudo deverá funcionar direitinho. Agora nós só precisamos compilar o projeto para testar.

Observe que as funções são bem simples, não possuindo qualquer espécie de crítica aos dados entrados. Apesar disto não ser recomendável para um programa profissional elas atenderão bem aos nossos propósitos que é de entender o funcionamento do Anjuta e como as funções possuem comentários entre as linhas de código também não irei falar sobre o funcionamento delas, bastando apenas que o leitor de uma lida no código fonte para entender seu funcionamento.

Qual comando usar para compilar?

Se você der uma olhada no menu "**Compilar**" verá que temos cinco comandos para compilação: **Compilar [F9]**, **Compilar com Make [SHIFT]+[F9]**, **Compilar [F11]**, **Compilar Tudo [SHIFT]+[F11]** e **Compilar Distribuição**. Mas quando nós vamos usar um e quando usar outro? A resposta para esta pergunta depende da situação e do tamanho do projeto. Veja só:

Comando "Compilar" [F9]: Gera apenas o arquivo-objeto (".o") equivalente ao arquivo atualmente aberto, se por acaso houver dependências você não terá sucesso na compilação, por isso dificilmente você usará este comando, já que a maioria dos projetos usam vários arquivos e bibliotecas separadas.

Comando "Compilar com Make" [SHIFT]+[F9]: Assim como o comando anterior apenas gera os "arquivos-objeto", com a diferença de que por usar os arquivos *makefile* ele pode compilar as dependências de arquivos e bibliotecas quando necessário. Este será o comando que você mais usará durante a fase de testes antes de gerar um executável. Cada vez que você alterar um arquivo do seu projeto tão importante quanto salvar será compilá-lo com esta opção para

atualizar o arquivo ".o" correspondente. Abuse do uso deste comando para fazer testes de sintaxe, quando houver necessidade, já que para gerar um código-objeto é necessário que ele cheque a sintaxe do seu código.

C**omando "Compilar" [F11]:** Este comando gera os arquivos ".o" e liga-os com as bibliotecas necessárias para gerar o nosso arquivo executável. Quando estamos lidando com um projeto pequeno podemos seguramente descartar as outras duas opções de compilação citadas antes e usar apenas este, assim agilizando o trabalho, mas quando o projeto envolve dezenas ou centenas de arquivos e bibliotecas ai a coisa muda de figura, pois se usar este comando diretamente você poderá esperar até vários minutos talvez até chegando a algumas horas esperando que todos os arquivos sejam recompilados, por isso não recomendo que você que está iniciando pegue esse mal vício.

Comando "Compilar Tudo" [SHIF]+[F11]: Este comando é similar ao comando explicado acima, mas com o diferencial de que enquanto aquele comando anterior compila apenas o que está no diretório de código fonte este desce até o "diretório raiz do projeto" e compila todos os arquivos a partir daí gerando assim um executável completo. Caso você tenha arquivos com código fonte em mais de um diretório será interessante usar esta opção para compilar realmente todo o seu projeto. Isso é comum no caso de se estar desenvolvendo um programa modular, onde você terá um diretório para cada módulo. Por exemplo um diretório com o módulo de controle de clientes outro com o módulo de controle de estoque etc. Este comando equivale ao famigerado "make all" já conhecido de muitos leitores que necessitam compilar programas de terceiros.

Comando "Compilar Distribuição": Com este comando você poderá criar um pacote ".tgz" contendo o código fonte do seu projeto.

Compilando e rodando o projeto

Para compilar o nosso projeto e verificar se há algum erro de sintaxe tecle **[SHIFT]**+**[F9]**, ou no menu **Compilar** escolha **Compilar com make** e em seguida, se tudo correu bem, tecle **[F11]** para gerar o arquivo binário e **[F3]** para executar e experimente realizar alguns cálculos com as quatro operações e veja se está realmente tudo ok.

Se durante a compilação ocorreu algum erro de sintaxe verifique se nada foi digitado errado, corrija e faça o procedimento citado acima. Lembrando ainda que como o nosso projeto é pequeno e só possui um arquivo de código nós podemos pular a compilação com make e apenas gerar o binário, que tudo deverá funcionar sem traumas.

Dividir para conquistar!

Imagine se o código fonte do OpenOffice.org, do Kernel Linux, ou do Gnome estivesse todo em um único arquivo. Seria algo terrível de se manter, talvez não houvesse memória suficiente para carregar todo o arquivo na maioria dos computadores para ser editado, a manutenção por múltiplos desenvolvedores, acredito eu que, seria inviável. Descer com o cursor até a linha 1.345.908.445 para retirar um simples "}" que havia sido esquecido ali então seria uma verdadeira tortura.

Por estes e outros motivos mais o código fonte de uma aplicação é dividido entre vários arquivos. As vantagens vão desde a manutenção de arquivos pequenos até a possibilidade de existir mantenedores para cada "grupo de arquivos" etc. A divisão de um projeto em vários arquivos depende de diversas ferramentas tais como o make, automake, autoconf etc que entre outras coisas verificam as dependências de bibliotecas e entre cada arquivo para que tudo funcione corretamente no ato da compilação.

Se tudo fosse tão fácil e simples, talvez eu nem estivesse escrevendo esta sessão e acredito até

que não haveria necessidade de estar fazendo um tutorial sobre o Anjuta. Mas ainda bem que o Sr. Naba Kumar foi iluminado ao desenvolver esta aplicação, por isso veremos detalhadamente como organizar a sua aplicação a partir daqui!

Criando os novos arquivos

A primeira coisa a se fazer para dividir o nosso projeto em vários arquivos é ter os demais arquivos com as extensões ".h" e ".c" para depois serem incluídos no código, via diretivas "#include".

Para criarmos os arquivos usamos o comando "**Novo**" do menu "**Arquivo**", digitamos um nome para o arquivo, escolhemos o tipo e as informações que desejamos, tais como cabeçalho e licença GPL. Ilustração 11 mostra a janela para criar um novo arquivo.

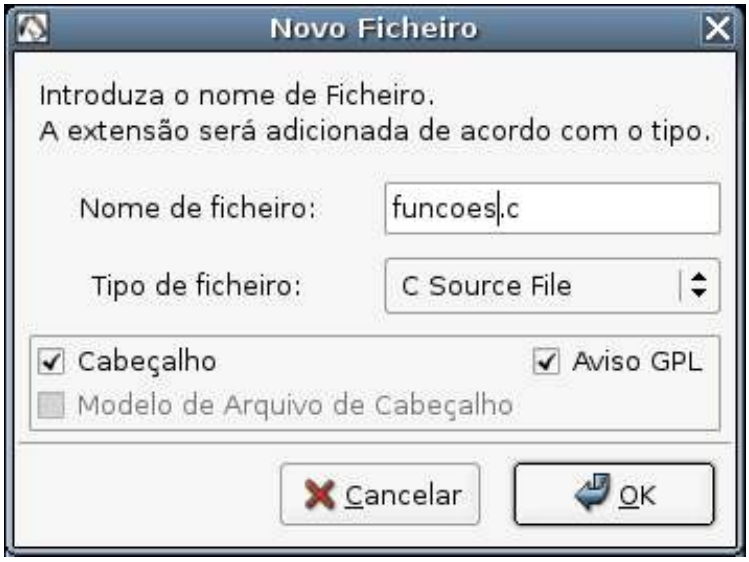

Ilustração 11 - Janela para criar um novo arquivo no Anjuta

Após digitar as informações clique em OK para que uma nova "aba" seja criada na área de trabalho do Anjuta. Nós realizamos este procedimento duas vezes: Uma para criar o arquivo ".c", onde terá todo o código das nossas funções e outra vez para criar o arquivo ".h" onde terá os protótipos, diretivas, macros e variáveis globais do nosso projeto. No menu "**Tipo de Arquivo**" da janela "**Novo arquivo**" isso equivaleria a escolher "C Source File" e em seguida "C – C++ Header File".

Uma vez que já temos os arquivos vazios nós devemos recortar o código dos protótipos (conteúdo do "Quadro 1") e cola-lo no arquivo ".h" e em seguida fazer o mesmo com o corpo das funções (conteúdo do "Quadro 2") no arquivo ".c". É importante frisar aqui que todo arquivo ".c" no nosso projeto deve permitir ser compilado sem dependências implícitas. Isso significa que o nosso arquivo "funcoes.c" deverá possuir uma diretiva "include" para cada uma das suas dependências. Em outras palavras nós devemos incluir as linhas a seguir logo no início do arquivo "funcoes.c".

```
#include <gnome.h>
#include <glade/glade.h>
#include "include/funcoes.h"
```

Quadro 3 - inclusões do arquivo "funcoes.c"

Com isso todas as funções do GTK+/Gnome e Glade serão encontradas nas duas primeiras

diretivas (Não é preciso ser adivinho para saber o que colocar aqui basta copiar o que está no arquivo “main.c”) e a terceira diretiva é para que o gcc possa encontrar onde está o respectivo arquivo “.h” complementar deste arquivo “.c”. A propósito esta terceira diretiva deverá entrar também no arquivo “main.c” pois este arquivo precisará conhecer as funções e variáveis que estão “*prototipadas*”/declaradas em “funcoes.h”

Após colar o conteúdo de cada arquivo salve-os mas observando o seguinte: Os novos arquivos “.h” e “.c” ainda não pertencem ao nosso projeto, por isso devem ser salvos em algum lugar fora do diretório de projeto (eu costumo salvá-los em /tmp). Depois eles serão adicionados ao projeto.

Vale aqui uma nota de que se mantivermos tudo em um arquivo “.h” o programa até será compilado sem muitas complicações mas isso acarretará no fato de que não poderemos compilar cada arquivo “.o” (arquivo-objeto) em separado fazendo com que todo o nosso trabalho de separar o projeto em pequenos arquivos sejam em vão, uma vez que não é permitido compilar arquivos “.h”.

Incluindo arquivos de código no projeto

Agora que o código está salvo em todos os arquivos nós vamos incluí-los no diretório de projeto. Isso é necessário para que fique tudo organizado. Para tanto vá ao menu **Projeto**, escolha o comando **Adicionar arquivo** e a opção **Arquivo de Inclusão**. Na janela que será exibida (como na Ilustração 12) navegue até o local onde você havia salvo os arquivos e selecione o arquivo “funcoes.h”.

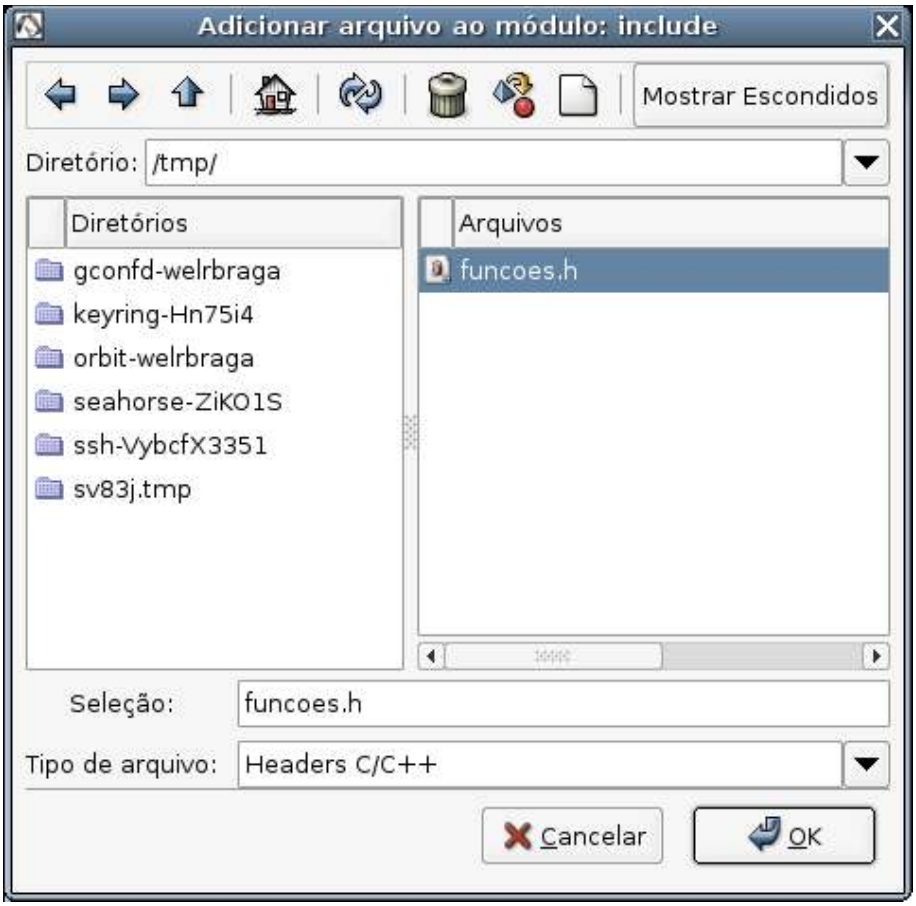

Ilustração 12 - Janela para adicionar Arquivo de Código ao projeto

Assim que você clicar o botão OK a janela da Ilustração 13 será exibida e você precisará apenas responder SIM. para que o arquivo seja copiado para a pasta adequada.

Ilustração 13 - Confirmação da inclusão do novo arquivo no projeto

Feito isso repita o mesmo procedimento descrito, mas desta vez usando a opção **Arquivo de Código Fonte**, para importar o arquivo ".c".

Gerando os novos arquivos Makefile

Uma vez que já temos os novos arquivos incluídos ao projeto nós devemos gerar os novos Makefiles e demais arquivos para que a compilação seja bem sucedida.

A primeira coisa a fazer é limpar todo o diretório de projeto usando o comando **Limpar Tudo** do menu **Compilar**. Isso fará com que todos os arquivos temporários, objetos makefiles etc sejam removidos; em seguida nós vamos usar o comando **Autogerar** do mesmo menu **Compilar** para que eles sejam recriados.

Se tudo foi feito corretamente então nós poderemos compilar o projeto e testar se ele ainda funciona. Lembrando que podemos usar apenas a tecla **[F11]** que se encarregará de compilar tudo, mas como nós precisamos testar se cada arquivo ".c" será compilado independente dos demais nós vamos selecionar primeiro o arquivo "funcoes.c" e usar o **[SHIFT]+[F9]**, em seguida faremos o mesmo para o arquivo "main.c" e só depois vamos gerar o binário com **[F11]**.

Se nenhuma mensagem de erro foi exibida ao teclarmos **[F3]** nossa calculadora ainda deverá estar funcionando.

Dividir, dividir e dividir

Um dos maiores objetivos da divisão de um projeto, como já foi dito, é organizar as suas funções por categoria, tipo de tarefa ou outros critério a gosto do desenvolvedor. Se nós olharmos para o nosso arquivo "funcoes.c" e "funcoes.h" vemos ai que há dois tipos de funções distintas: As funções que funcionam como manipuladores de sinais "on_buttonN_clicked()" e as funções de uso geral "get_valor()" e "set_valor()" que respectivamente lêem ou definem o valor das caixas de entrada e rótulo de resultado.

Para fixarmos o procedimento e vermos que esta tarefa não é tão difícil como deve ter parecido ao fazermos pela primeira vez, nós faremos novamente, para separar desta vez estes grupos de funções.

1 – A primeira coisa a fazer é criar os arquivos "funcoes_geral.c" e funcoes_geral.h", usando para isso o menu **Arquivo**, comando **Novo**. Quando exibir a janela "Novo Arquivo", nós digitamos o nome do arquivo e escolhemos o seu tipo, conforme já comentado no tópico "Criando os novos arquivos"

2 – Agora nós iremos recortar as funções "get_valor()" e "set_valor()" que estão no arquivo "funcoes.c" e colá-las em "funcoes_geral.c". Fazendo o mesmo com os seus protótipos que estão em "funcoes.h", mas desta vez, colando-os em "funcoes_geral.h". Observe ainda que a variável "xml" deverá também ser movida para este novo arquivo. E o por que disso é simples de ser explicado: Se você olhar o código fonte do arquivo "funcoes.c" verá que ele não possui qualquer referência a esta variável, em contra partida se olhar o novo arquivo "funcoes_geral.c" verá que

ambas as funções deste arquivo necessitam desta variável. Note ainda que o arquivo "funcoes_geral.c" necessita do mesmo cabeçalho que está no arquivo "funcoes.c", isto é, a inclusão dos headers "gnome.h" e "glade.h, já que ele também usa funções prototipadas nestes arquivos. E pra completar o arquivo "main.c" necessita incluir o cabeçalho deste novo arquivo também, já que a variável "xml" está ali. Se isto não for feito na hora de compilar este arquivo ocorrerá um erro acusando que esta variável não foi declarada.

3 – Ainda assim se compilarmos o projeto teremos um erro de compilação, pois as funções "get_valor()" e "set_valor()" não serão encontradas. Por que elas não serão encontradas? Simples, apesar de termos salvo os arquivos nos respectivos diretórios "include" e "src" eles ainda não fazem parte do projeto, ou seja, ainda não estão declarados nos arquivos "makefile". Para incluirmos no projeto temos que usar o comando "**Adicionar Arquivo**" do menu "**Projeto**", uma vez para cada arquivo (o arquivo de código ".c" e em seguida o arquivo de header ".h"). Depois de feito isso, ai sim poderemos compilar o projeto que tudo será compilado corretamente.

Onde eu errei?

A pior tarefa durante o processo de desenvolvimento de uma aplicação é a depuração. O Anjuta possui uma integração muita boa com o GDB (GNU Debugger) o que permite usar todo o poder desta ferramenta sem precisarmos sair do ambiente do Anjuta. A depuração constitui-se basicamente em seguir o processamento de uma aplicação passo-a-passo para acompanharmos como as variáveis estão se alterando ao longo do programa.

Imagine a seguinte situação: Sua aplicação tem uma função que recebe um valor em formato "gchar" para internamente converter em "gint" e só em seguida calcular o fatorial deste valor. Só que ao exibir o resultado o que se tem é lixo, ou seja, um valor que não tem nada a ver com o esperado. Pode existir aqui várias hipóteses para este problema, como por exemplo você está calculando o fatorial de forma errada, ou a rotina de conversão de "gchar" para "gint" pode estar funcionando mal etc. Mas como descobrir o problema?

Com a sessão de depuração aberta nós poderemos executar o programa e faze-lo "suspender" na última linha da função, por exemplo, e com isso analisar todas as variáveis locais em busca de uma atribuição inválida.

Apesar do nosso programa não conter erros nós vamos executar a depuração para ver como usar este recurso para depurar um programa.

A primeira ação a se tomar para usarmos o depurador GDB no Anjuta é iniciar a sessão de depuração a partir do menu **Depurar** com o comando **Iniciar Depurador** ou teclar **[SHIFT]**+ **[F12]**. Ao fazer isso na aba "Debug" da nossa "janela de mensagens" aparecerá a mensagem: "*O depurador está preparado*".

Isso significa que o Anjuta conseguiu carregar o "servidor do GDB" que será responsável por processar as nossas solicitações de depuração. Agora nós poderemos acompanhar a execução do nosso código passo-a-passo e quando terminarmos basta usar o comando **Parar o Depurador** do mesmo menu **Depurar**.

Marcando os pontos de interrupção

Agora que o depurador já está rodando nós devemos definir um ou mais pontos de interrupção. A finalidade deles é de informar ao depurador onde o programa deverá ser suspenso para análise.

Uma das formas de marcarmos os pontos de interrupção é usando o menu **Depurar**, comando **Pontos de interrupção** e a opção **Definir pontos de interrupção**, mas há maneiras mais

simples de se fazer a mesma tarefa como por exemplo um duplo clique sobre a "margem marcadores", logo a frente da linha desejada. Veja na Ilustração 14. Para desmarcar um ponto de interrupção, desativá-lo ou simplesmente limpar todos eles use os respectivos comandos no menu **Depurar**, comando **Pontos de Interrupção**.

Nota: A Margem de marcadores pode ser ativada/desativada clicando sobre o editor de textos com o botão direito do mouse, escolhendo **Opções** e em seguida clicando em **Alternar margem de marcador**.

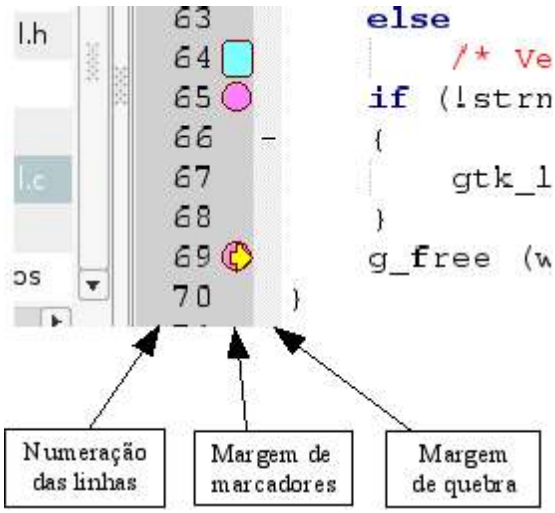

Ilustração 14 - Margens da janela de edição

Para esta demonstração eu marquei uma interrupção na primeira linha de código da função set_valor(), no arquivo funcoes_geral.c, ou seja, a linha que contém o código:

```
GtkWidget *wid = glade_xml_get_widget (xml, meu_widget);
```

Quadro 4 – Primeira linha de código da função set_valor(), onde foi marcada a interrupção

Uma vez que o depurador esteja ativado nós vamos rodar o programa usando a tecla **[F4]**, ou o menu **Depurar**, comando **Execução**, e opção **Executar/Continuar** que permitem executar o nosso programa com os recursos de depuração.

Uma vez que o nosso programa esteja sendo executado nós usaremos as "abas": *Debug, Locals, Stack* e *Watchs* para avaliar a situação atual e controlaremos o programa usando as teclas de **[F4]** até **[F8]**.

Informações disponíveis durante a depuração

A aba **debug** mostra informações em geral sobre o depuração, tal como o número do "breakpoint", chamada da função e seus parâmetros, arquivo onde a função se encontra e número da linha entre outras informações.

Por exemplo:

```
Preparando para iniciar a sessão de depuração ...

Carregando Executável: /
home/welrbraga/Projetos/HelloWorldAnjuta/src/hellowordanjuta
Using host libthread_db library "/lib/tls/libthread_db.so.1".
O depurador está preparado.
```

```
Executando o programa ...
[Thread debugging using  libthread_db enabled]
[New Thread 1088870080 (LWP 24901)]
[Switching to Thread 1088870080 (LWP 24901)]
main (argc=1,argv=0xbffffbd4) at main.c:25

Breakpoint 1, set_valor(meu_widget=0x98678bd "label1", valor=6)
em funcoes_geral.c:49
```

Quadro 5 – Saída da aba "debug" quando o depurador estiver ativo, logo após atingir um breakpoint

A aba **Locals** mostra as variáveis locais da função atual com seus respectivos valores. Esta aba será bastante útil se você estiver analisando o comportamento das variáveis locais de determinada função. A cada vez que uma variável mudar de valor ela será destacada com a cor vermelha.

Pra completar, ao clicarmos com o botão direito do mouse sobre qualquer uma das variáveis locais nós poderemos mudar o seu formato de exibição entre Formato padrão, Binário, Octal, Decimal com sinal, Decimal sem sinal, Hexadecimal, Caracter ou ainda inspecionar a região de memória onde o dado encontra-se armazenado.

Por exemplo:

Variável	*Valor*
wid	(GtkWidget *) 0x40000000
wid_tipo	(gchar *) 0x0
buffer	"\230a\021\b"
valor_txt	(const gchar *) 0x8949105 "Fgfdg46gfdbfg\045f4$\01234FGRE\234\231FGRFRy6546y"

A aba **Stack** mostra a pilha de chamada de funções desde a função "main" no seu programa principal, passando por todas as funções nas bibliotecas vinculadas até a função que está sendo depurada.

Clicando com o botão direito sobre qualquer uma das linhas podemos ver informações referentes ao estado dos registradores e memória neste instante de tempo em que o programa parou. Se for necessário ver do instante anterior (chamada de função anterior) basta dar um duplo clique sobre ela, antes.

Exemplo:

Ativo	*Contagem*	*Frame*
=>	0	`set_valor(meu_widget=0x98678bd "label1", valor=6) em funcoes_geral.c:49`
	1	`0x08048fbfd8 in on_button3_clicked() at funcoes.c:42`
	2	`0x408633b6 in g_cclosure_marchal_VOID_VOID() from / usr/lib/libgobject-2.0.so.0`
	...	`...`
	31	`0x4045bc83 in gtk_main() from /usr/lib/libgtk-x11-2.0.so.0`
	32	`0x08048ee9 in main (argc=1,argv=0xbffffbd4) at main.c:48`

A aba **Watches** é permite inspecionar qualquer variável no seu programa. Desde que ela esteja dentro do seu escopo. Isso significa que se uma variável local estiver na lista de inspeção, quando você sair desta função a variável deixará de ter seu valor visível. Ainda assim é uma ferramenta útil, e muito útil, já que você poderá definir aqui apenas as variáveis que realmente devem ser monitoradas, sendo elas locais ou globais e ainda pode realizar algumas operações com elas se houver necessidade: Como por exemplo fazer atribuições, realizar cálculos aritméticos com as variáveis fazer verificações lógicas etc.

Para que se possa adicionar novas variáveis ou expressões a serem inspecionadas clique com o botão direito do mouse em qualquer local nesta aba e escolha a opção “Adicionar expressão”. O mesmo é válido se você quiser remover uma variável da lista ou limpar toda a lista.

Ex.:

Variável	***Valor***
valor	= 27
valor_txt	= (const gchar *) 0x8049105 “fgert\34fdfhgyutygk$%$\78dfsd\23\45\123”
buffer	= “8\017\016\b”
buffer[0]	= 56 '8'
buffer[1]	= 15 '\017'
buffer[2]	= 14 '\016'
buffer[3]	= 8 '\b'
buffer[4]	= 0 '\0'
buffer[1]==buffer[2]	= 0

Controlando seu 'elefantinho'

Como já comentado antes as teclas de função de **[F4]** até **[F8]** são usadas para controlar a sua aplicação em tempo de depuração.

Se você já ativou o depurador e executou o programa com a tecla **[F4]** notará que ele estará funcionando 'quase' normalmente. Eu disse quase por que ao clicar em um botão de operação ele parece ter congelado. Mas não parece não, ele realmente foi congelado para que você pudesse analisar a situação. Agora você precisa conhecer a função de cada uma das teclas usadas para controle da aplicação, pois senão você vai acabar reiniciando o computador, achando que tudo está perdido e o mundo acabou!

Tecla **[F4]** – (Menu **Depurar**, Comando **Execução**, opção **Executar/Continuar**) – Esta tecla é usada para executar o programa, fazendo-o interromper a cada 'breakpoint' que for encontrado. Se o seu programa já parou em um 'breakpoint' e você quer fazê-lo prosseguir normalmente até o próximo 'breakpoint', ou até o final do processamento, caso não haja mais breakpoints pelo caminho, basta tecla-la novamente.

Por exemplo, quando nós temos uma função que manipula algumas variáveis e queremos ver o que acontece com elas no decorrer do processamento basta marcarmos um 'breakpoint' no início da função e outro no final. Ao teclarmos **[F4]** pela primeira vez o programa será executado normalmente até o primeiro 'breakpoint', quando será suspenso e teremos o estado inicial das variáveis. Após vermos a situação delas, teclamos **[F4]** novamente que o processo seguirá até atingir o outro 'breakpoint' no final da função. Quando isso ocorrer o programa entra em suspensão novamente e as variáveis alteradas serão destacadas com cor vermelha, exibindo seus

novos valores.

Tecla **[F5]** – (Menu **Depurar**, Comando **Execução**, opção **Passo para dentro**) – Esta tecla é usada para avançar linha-a-linha dentro do programa, se chegar a uma linha que chame uma outra função sua você será levado automaticamente até a primeira linha desta função, quando chegar no final dela e ainda pressionar [F5] você será automaticamente levado a função anterior de onde esta foi chamada até que não haja mais qualquer linha a ser executada.

Obs: Quando chegar na última linha de código possível de ser executada, mesmo que se pressione novamente a tecla **[F5]** a sua aplicação continuará congelada. Tecle **[F4]** para continuar a execução normalmente.

Por exemplo exemplo, se houver um "breakpoint" na linha: `gdouble e2 = get_valor ("entry2")` - segunda chamada a esta função no manipulador on_button3_clicked - ao teclar **[F5]** você entrará na função "get_valor()" podendo analisar linha a linha desta função. É por isso que esta ação se chama "passo para dentro".

Tecla **[F6]** - (Menu **Depurar**, Comando **Execução**, opção **Passo por cima**) – Esta tecla fará com que a depuração avance linha-a-linha, assim como a tecla [F5], mas com a diferença de que havendo uma chamada de função no caminho, como no exemplo citado antes, o depurador passará por cima dela, ou seja, irá executar aquela função normalmente como se fosse um comando da linguagem e avançará para a linha seguinte da sua função.

Tecla **[F7]** - (Menu **Depurar**, Comando **Execução**, opção **Passo para fora**) – Aproveitando o exemplo dado anteriormente, nós puséssemos um "breakpoint" na função "on_button3_clicked", justamente na linha que chama a função "get_valor()", mas nós não queremos entrar nesta função, então deveríamos pressionar [F6], para passar por cima de "get_valor", mas por um descuido pressionamos [F5] indo diretamente para a primeira linha desta função ... a função "get_valor()" é grande demais para avançarmos linha-a-linha e cancelar a depuração para recomeçar novamente o fará perder muito. A solução para isso é a tecla [F7] que dará um passo para fora de get_valor, indo para a linha seguinte da função que estávamos antes, como se nada de errado tivesse acontecido.

Tecla **[F8]** - (Menu **Depurar**, Comando **Execução**, opção **Executar até o cursor**) – Esta tecla simplesmente executa o seu código até a posição atual do cursor. Isso é útil quando você não quer se dar ao luxo de marcar apenas um "breakpoint". Basta clicar na linha desejada e teclar [F8].

Conclusão

Isso é apenas uma parte do "tudo que se pode fazer" com o Anjuta. Eu não abordei o uso do CVS que é um poderoso recurso para controle de versões e é interessantíssimo para projetos que serão mantidos por uma equipe de desenvolvedores, onde todos poderão mexer no código fonte, sem no entanto perderem as alterações implementadas pelos demais participantes do grupo. Talvez num futuro próximo eu venha a escrever sobre este recurso e até mesmo incluí-lo neste mesmo documento, mas por enquanto é só.

Acredito que daqui para frente seja possível ao leigo aspirante a desenvolvedor GTK/Glade criar seus programas com uma interface amigável, agradável e prática.

Obrigado pela leitura e espero que tenha gostado. Dúvidas, sugestões, elogios e críticas podem ser feitas a partir do site.

Felicidades e boas linhas de código!

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

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. Secondarily, 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.