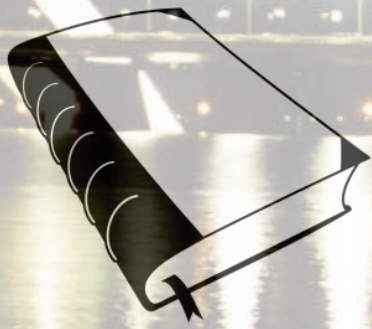
Edição 3

revista Qt

Janeiro-Fevereiro/2011



Primeiro capítulo do livro: "Como pensar como um cientista da Computação"

Criando um instalador para suas aplicações com o InstallJammer

Aula 1
Minicurso de programação C++ com Qt

Code less.
Create more.
Deploy everywhere.

Pés no chão, cabeça nas alturas...

Ah, os anos 80...
Separando os psicodélicos anos 70 dos monótonos anos 90.
Esta música do Ricardo Graça Melo foi tema do filme Garota Dourada, que tinha Marina - que depois passou a ser conhecida como Marina Lima - no elenco.
O filme era um romance despretensioso para adolescentes. A música... bom meio fraquinha mas eu gostava na época e... tá bom, eu ainda gosto um pouco.
O fato é que cada vez que ouço a expressão "Cloud computing" ou "Computação nas nuvens" me vêm à mente o começo daquela música: "Pés no chão, cabeça nas alturas...". Não é fácil.

No que se refere à tal da computação nas nuvens, concordo com a parte da cabeça nas alturas, desde que mantidos os pés no chão. Ter dados como emails, textos, planilhas, fotos, etc. espalhados por servidores web, é algo inevitável, mas daí até abrir mão de ter as minhas aplicações e dados salvos na minha máquina existe uma distância enorme.

Em uma época de grandes inovações, é natural que surjam "falsos profetas tecnológicos", pregando conceitos que nos remetam a verdadeiras revoluções. É uma forma de conseguir a atenção para um assunto.

Seja bem vinda a evolução, mas certas coisas não precisam mudar tão radicalmente.

Nem tanto ao mar, nem tanto à terra...

André Vasconcelos

Índice

Editor André Luiz de Oliveira Vasconcelos

Colaboradores Adriano H. Hedler
Rafael Fassi
Thiago Rossener Nogueira
Fórum QtBrasil

Minicurso de Programação C++ com Qt

"Quanto mais a gente ensina,
mais aprende o que ensinou."

- Roberto Mendes & Jorge Portugal

Uma frase adequada para definir a filosofia da Revista Qt e a minha postura em relação ao pouco conhecimento ao qual tenho acesso. Compartilhando conhecimento, ao mesmo tempo em que exercitamos, estendemos a outros a oportunidade de aprender.

Confirmando esta orientação da Revista Qt, estou publicando a partir desta edição, as aulas de um minicurso de programação em C++ com Qt.

Voltado àqueles que queiram utilizar o Qt, conheçam lógica de programação, mas não conheçam a linguagem de programação C++, este curso abordará desde os aspectos fundamentais da programação nesta linguagem, passando por programação orientada a objetos até a criação e manutenção de projetos C++ com Qt.

Para que tenhamos um melhor aproveitamento, as aulas terão um escopo mínimo e bem definido. Assim, cada assunto abordado será esgotado através de vários exemplos, demonstrações e exercícios. A ideia é que tenhamos ao final de cada aula, o domínio do assunto objeto da mesma.
É bom destacar que este é um "mini" curso, e portanto não tem a pretensão de cobrir TODOS os aspectos relacionados à programação em C++, à orientação a objetos ou ao Qt. A ideia é deixar os "alunos" (se me permitem chamá-los assim) prontos para passar para um próximo nível de aprendizado, no qual a documentação do Qt e a experiência com a ferramenta sejam o bastante.

Este curso terá basicamente três fases, a saber:

- Introdução à programação em C++
- Orientação a Objetos com C++
- Desenvolvimento com Qt

Nas duas primeiras fases são dedicadas exclusivamente ao aprendizado de C++, portanto não teremos contato algum com o Qt. Na última fase do curso, com o conhecimento de C++ adequado, passaremos ao desenvolvimento com Qt.

Como eu costumava dizer quando dava aulas de informática lá pelos anos 90, não quero prosseguir com o curso enquanto exisitirem dúvidas sobre o assunto de cada aula. Então não vou definir um roteiro para o curso, ou definir prazos para o mesmo. Conto com o retorno dos alunos (por email) para ir ajustando o tempo e o conteúdo do curso. Para que possamos compartilhar as eventuais dúvidas, sugestões, etc, criei um grupo específico para este minicurso. Anote o endereço e o email do grupo:

https://groups.google.com/group/professorvasconcelos?hl=pt-br

professorvasconcelos@googlegroups.com

Vamos combinar mais uma coisa: este canal - o grupo do Minicurso - deve ser usado apenas para questões relacionadas às aulas publicadas, ok?

Para esta primeira fase no nosso curso, vamos precisar instalar e configurar em nossas máquinas o ambiente para desenvolvimento em C++. O compilador adotado será o GCC para as plataformas Windows e Ubuntu Linux.

Quem for utilizar o sistema operacional Windows precisa fazer o download do MinGW através do endereço:

http://sourceforge.net/projects/mingw/

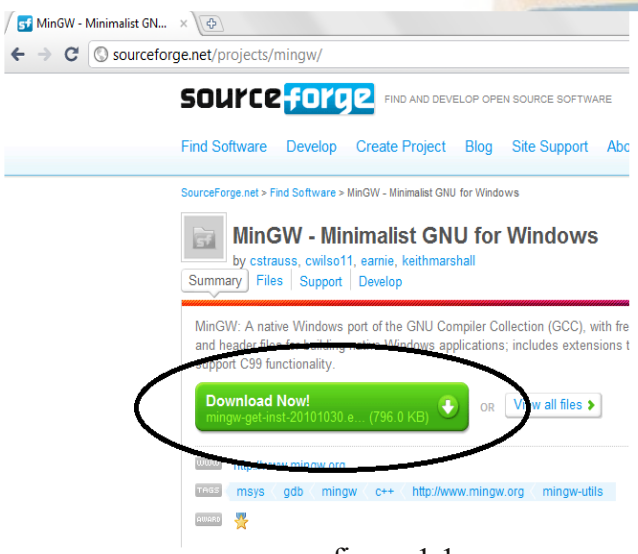


figura 1.1

Quando concluir o download, execute o arquivo de instalação do MinGW. A primeira tela do instalador é mostrada na figura 1.2:

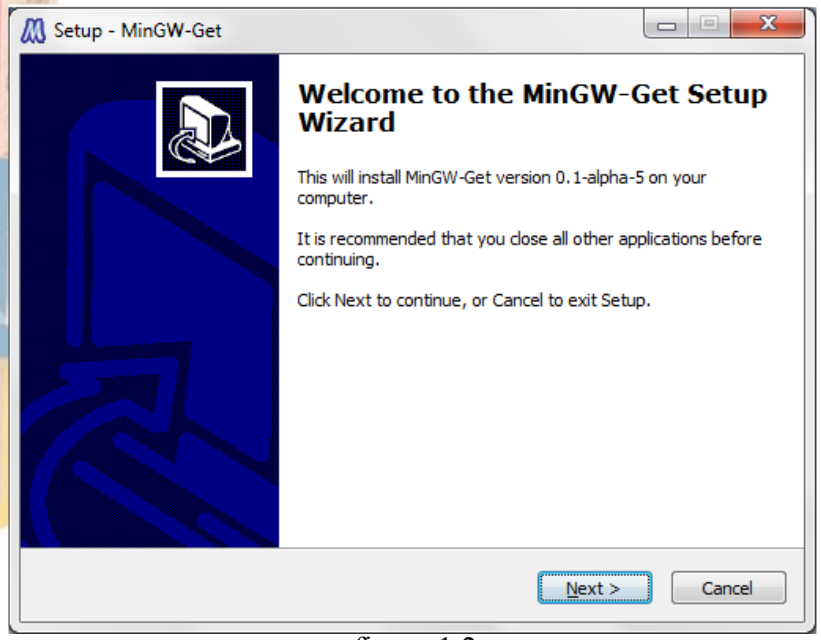


figura 1.2

Clique no botão Next para continuar. A próxima tela apresenta ao usuário o aviso de que é necessário ter permissão de administrador para prosseguir com a instalação do MinGW (v. figura 1.3).

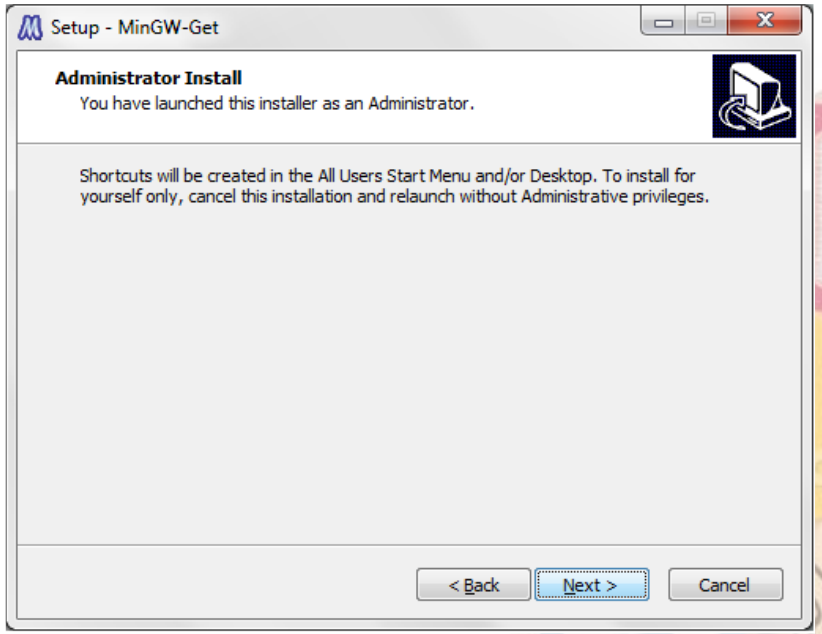


figura 1.3

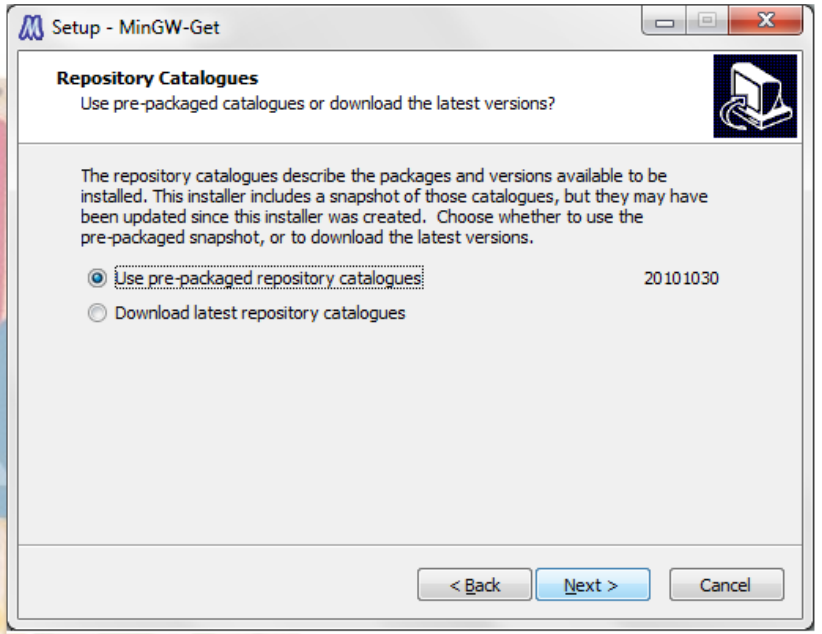


figura 1.4

O próximo passo na instalação permite que se instale versões mais atualizadas quando disponíveis. Como acabamos de baixar o instalador, a versão selecionada já corresponde à mais recente. Para continuar apenas clique no botão Next (figura 1.4).

Na próxima tela (figura 1.5), temos a licença de uso do MinGW. Para prosseguir marque a opção "I accept the agreement", indicando que você concorda com os termos da licença e clique novamente no botão Next.

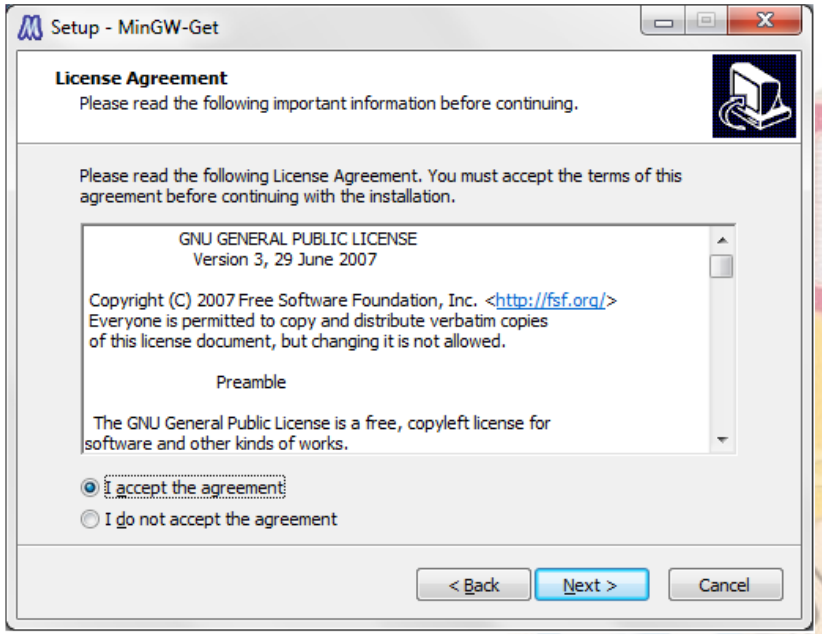


figura 1.5

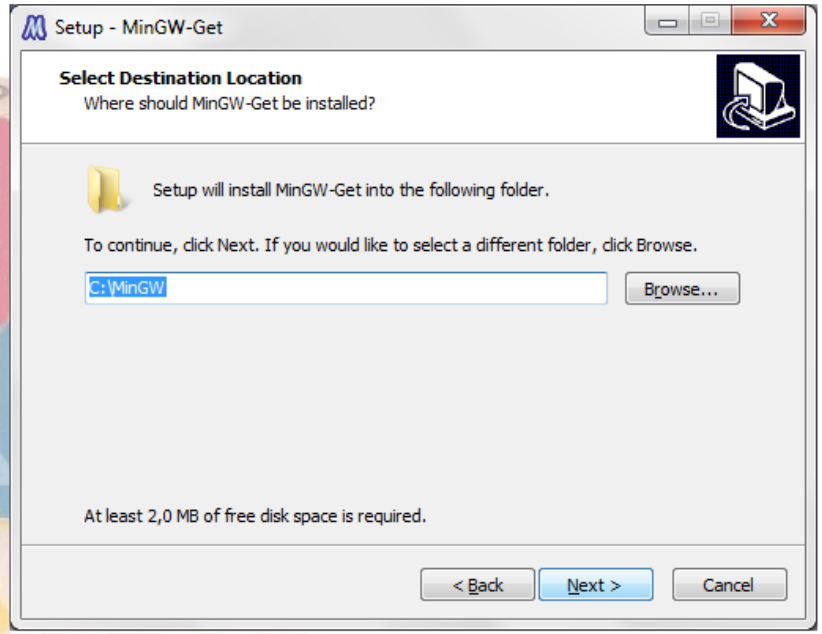


figura 1.6

O próximo passo é indicar ao instalador em qual diretório deverá ser instalado o MinGW. O diretório sugerido pelo instalador é o C:\MinGW. Quando a instalação for concluída, precisaremos desta informação para completar a configuração do nosso ambiente de desenvolvimento. Mais uma vez, clique no botão Next (figura 1.6).

Na tela mostrada na figura 1.7, o usuário pode indicar um nome a ser usado no menu Iniciar do Windows para a pasta do MinGW. O nome sugerido pelo instalador é MinGW mesmo. Caso não queira que o instalador crie uma opção para o MinGW no menu Iniciar do Windows, marque a opção "Don't create a Start Menu folder".

Para prosseguir com a instalação clique no botão Next.

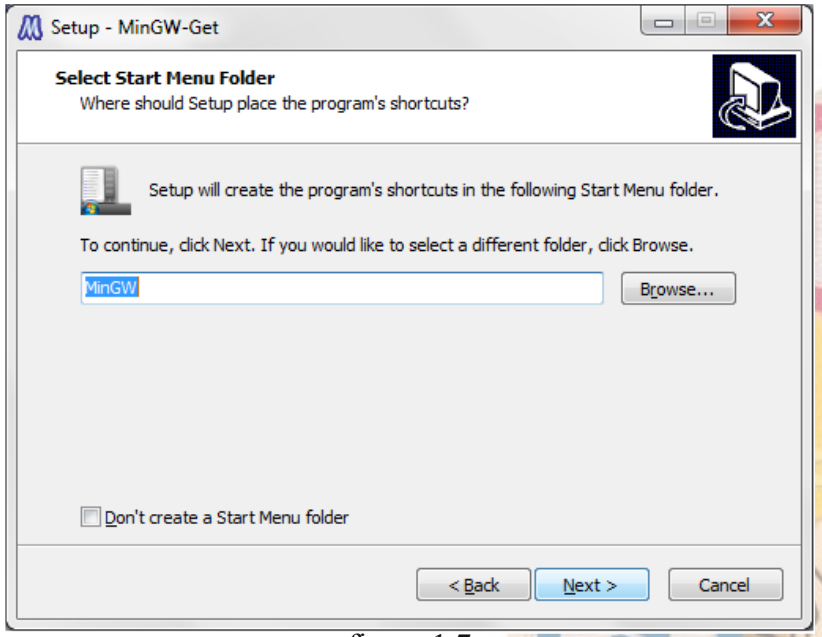


figura 1.7

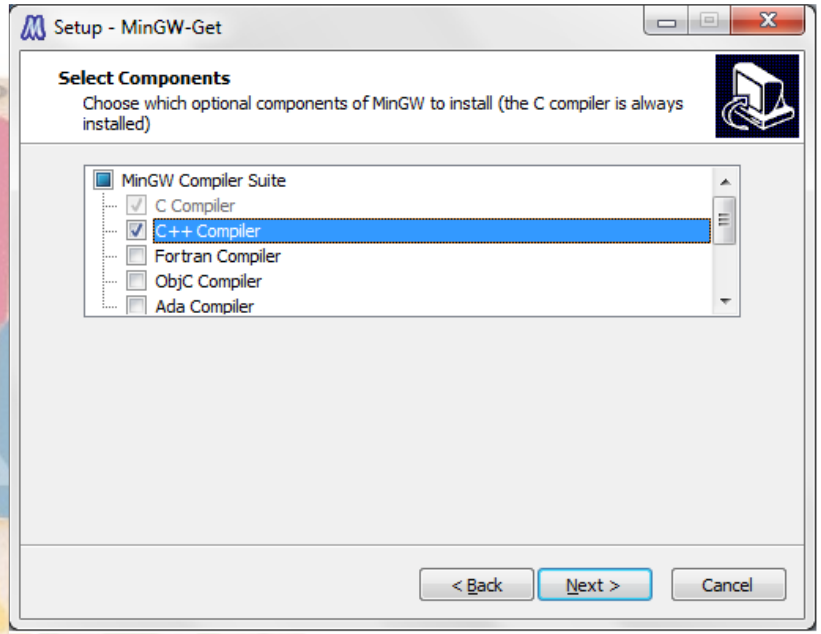


figura 1.8

O próximo passo da instalação permite que você selecione componentes a serem instalados. Agora você vai precisar fazer uma coisa antes de clicar no botão Next. Marque a opção C++ Compiler, que é o motivo pelo qual estamos instalando o MinGW e depois... adivinha... clique no botão Next (figura 1.8).

Finalmente o instalador nos apresenta um resumo das opções selecionadas nas telas anteriores e está pronto para iniciar a instalação propriamente dita, criando a estrutura de diretórios do MinGW com seus respecivos arquivos. Para iniciar o processo, clique no botão Install (figure 1.8).

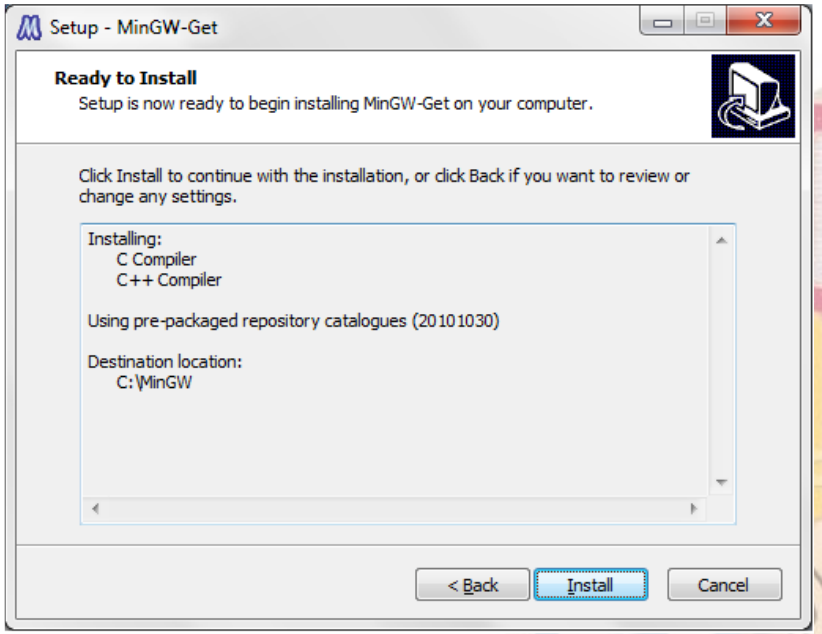


figura 1.9

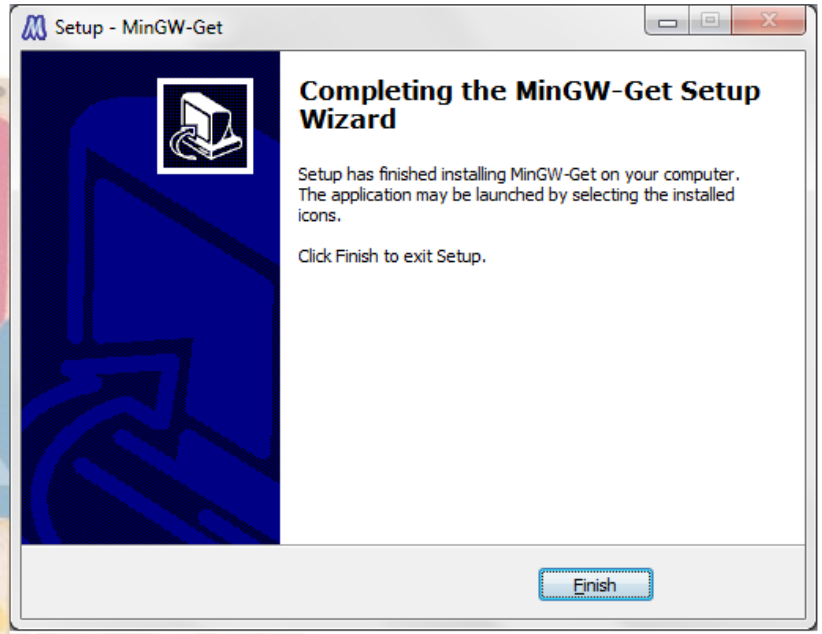


figura 1.10

Durante o processo, o instalador exibirá algumas informações como a cópia e download de arquivos. Como serão feitos downloads de alguns arquivos, a velocidade da sua conexão com a Internet vai acabar determinando o tempo da instalação.
Ao final do processo, será apresentada a tela mostrada na figura 1.10.

Pronto. O MinGW está instalado em sua máquina. Se você aceitou a sugestão do instalador, o MinGW foi instalado no diretório C:\MinGW. O compilador C++ que vamos usar em nosso curso, encontra-se no diretório bin, dentro do diretório MinGW (figura 1.11).

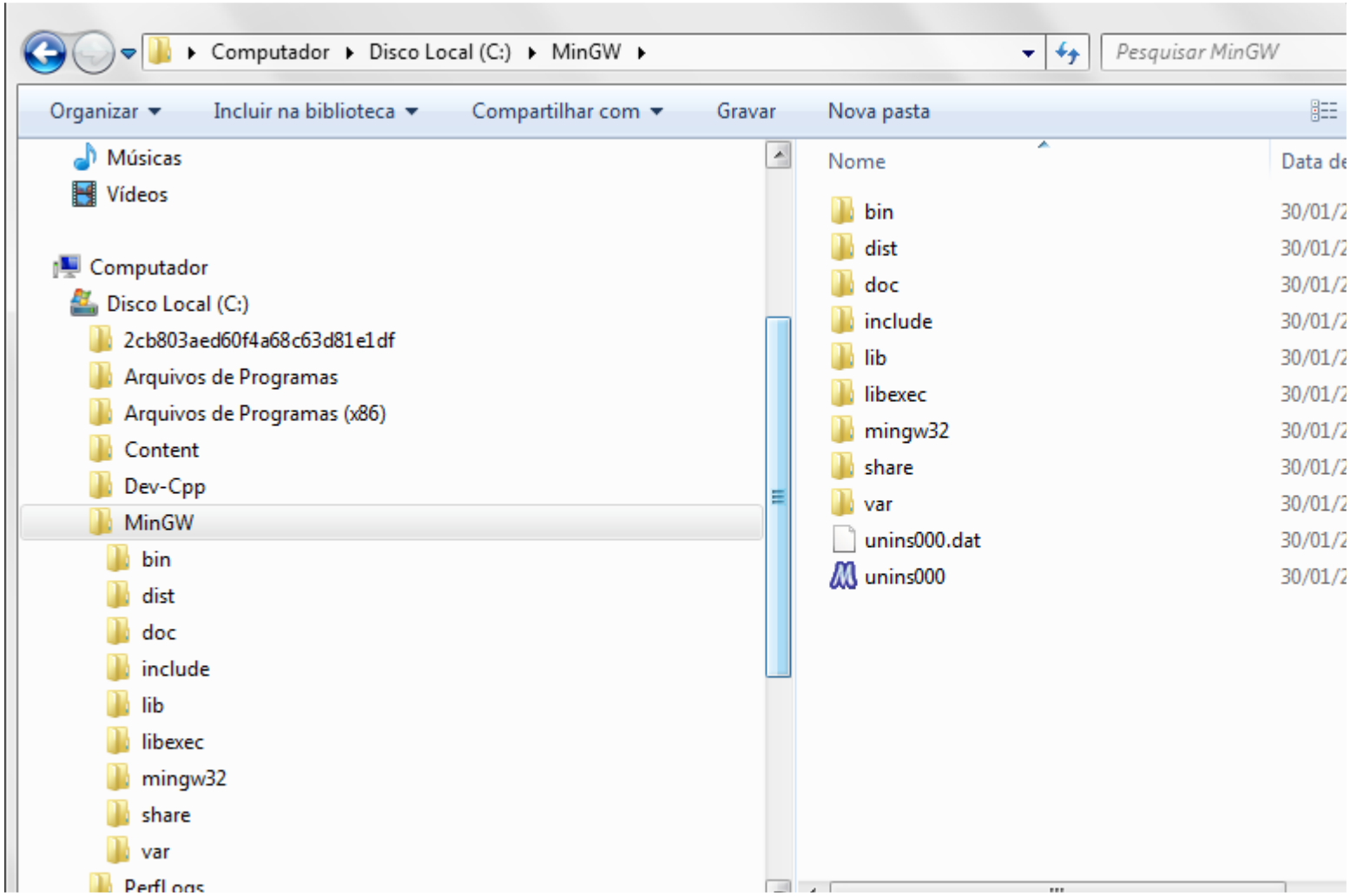

Computador ▸ Disco Local (C:) ▸ MinGW ▸
Pesquisar MinGW
Organizar
Incluir na biblioteca
Compartilhar com
Gravar
Nova pasta
Músicas
Vídeos
Computador
Disco Local (C:)
2cb803aed60f4a68c63d81e1df
Arquivos de Programas
Arquivos de Programas (x86)
Content
Dev-Cpp
MinGW
bin
dist
doc
include
lib
libexec
mingw32
share
var
PerfLogs
Nome
Data de
bin 30/01/2
dist 30/01/2
doc 30/01/2
include 30/01/2
lib 30/01/2
libexec 30/01/2
mingw32 30/01/2
share 30/01/2
var 30/01/2
unins000.dat 30/01/2
unins000 30/01/2

figura 1.11

Agora você precisa indicar ao Windows, o caminho para o compilador C++ do MinGW. Fazemos isso, alterando o conteúdo da variável de ambiente PATH. Para ter acesso às variáveis de ambiente do Windows, clique no menu Iniciar, selecione Painel de Controle -> Sistema -> Configurações avançadas do sistema. Na janela "Propriedades do Sistema", clique no botão Variáveis de Ambiente (figura 1.12).

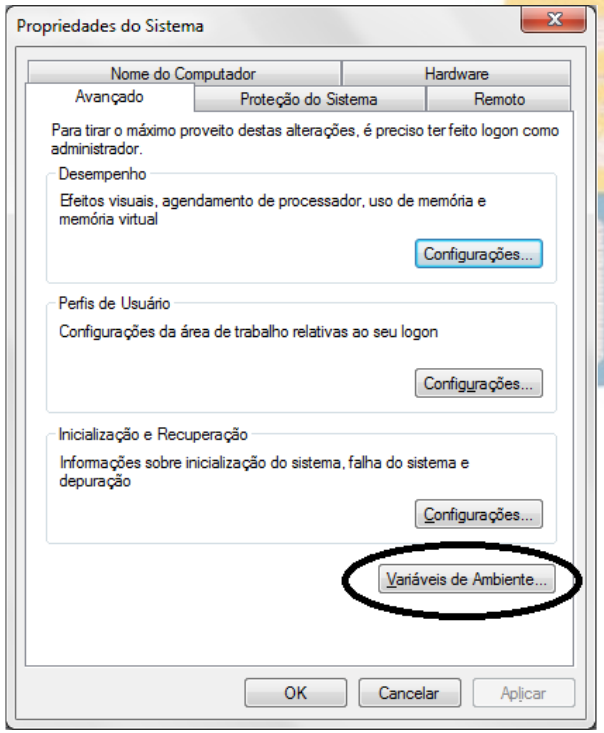


figura 1.12

Na janela "Variáveis de Ambiente", selecione la variável Path, na seção "Variáveis do Sistema" (v. figura 1.13) e clique no botão Editar.

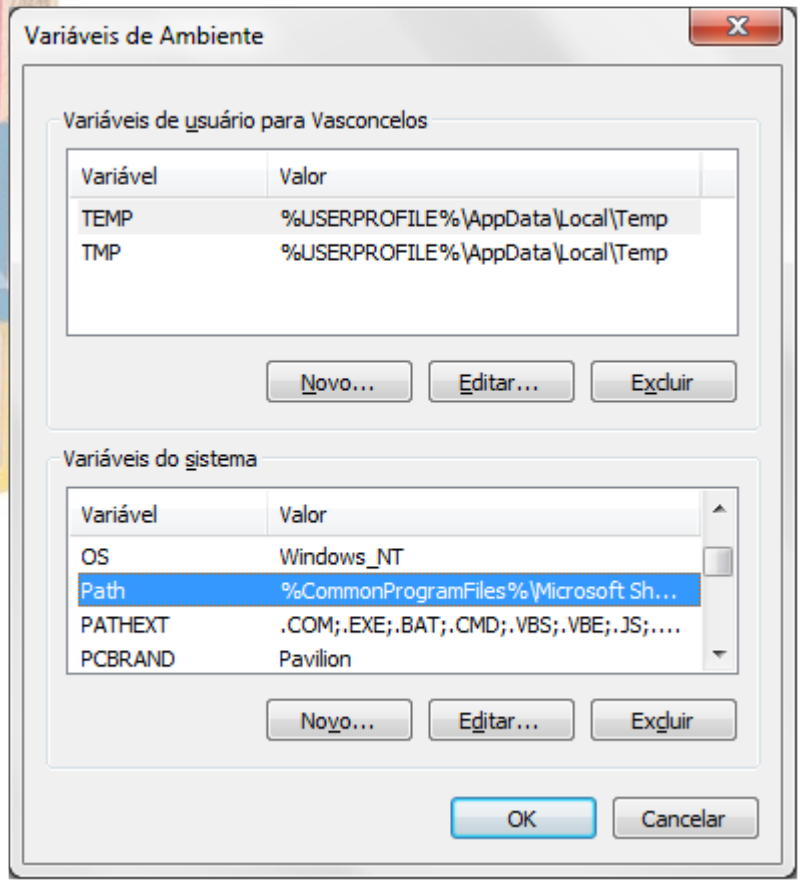


figura 1.13

Acrescente ;C:\MinGW\bin ao conteúdo do campo valor da variável (v. figura 1.14).

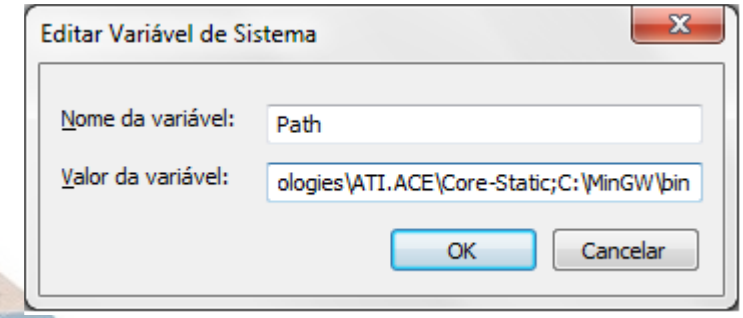


figura 1.14

Clique no botão Ok e... pronto! Para saber se funcionou, abra uma janela do Prompt de Comando pela opção: Iniciar -> Acessórios -> Prompt de Comando e digite o comando mingw32-c++, Se tudo correu bem na instalação e configuração da variável de ambiente Path, o sistema operacional deverá emitir a mensagem: mingw32-c++: no input files, como mostra a figura 1.15.
Isto encerra a configuração do nosso ambiente de desenvolvimento C++ no Windows.

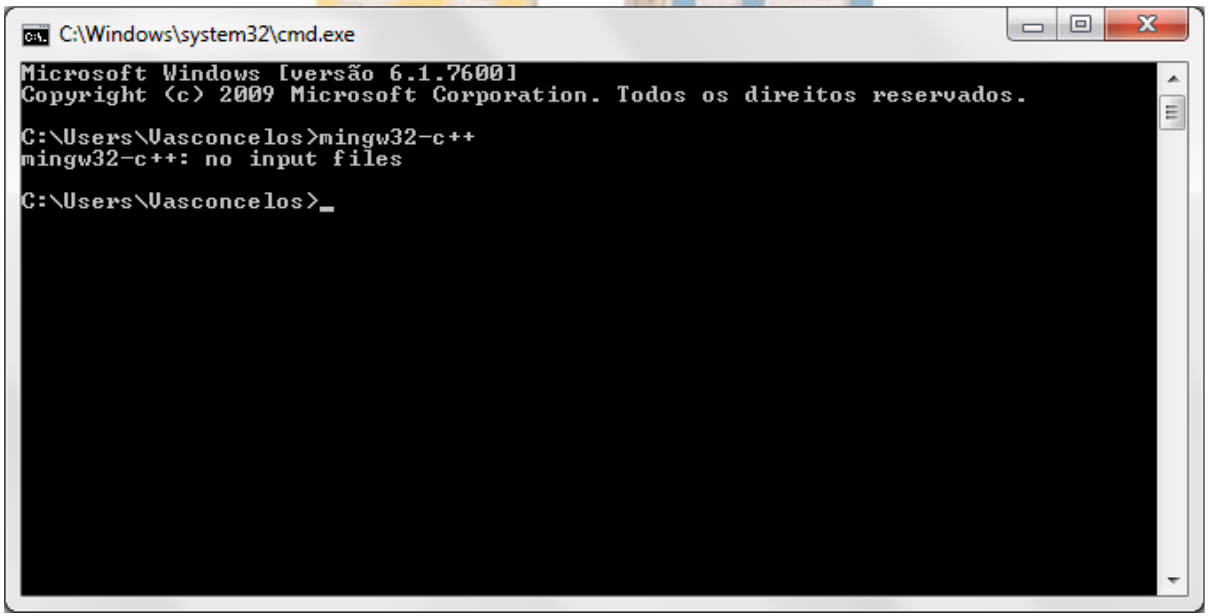


figura 1.15

Agora é a vez do Ubuntu Linux. A instalação do ambiente de desenvolvimento em C++ aqui será mais simples do que no Windows. Basta executar o comando a seguir:

$ sudo apt-get install build-essential

Você precisará ter a senha de root para executar esta operação. Para verificar se a instalação ocorreu com sucesso, você pode executar o comando g++, como mostra a figura a seguir:

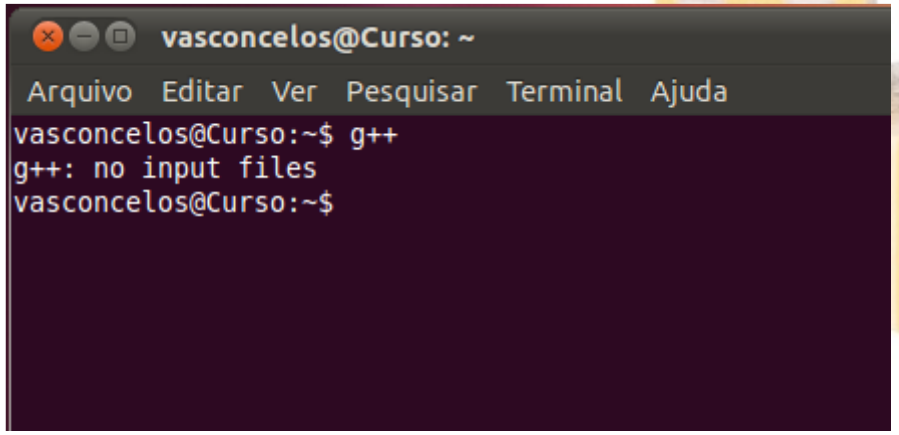


figura 1.16

Se você obteve o resultado acima, então o compilador C++ e as bibliotecas de desenvolvimento em C e C++ foram devidamente instalados. A mensagem "g++: no input files" indica que você executou o compilador sem indicar um arquivo a ser compilado. Não se preocupe com isso agora. No final desta aula veremos como compilar um programa.

Bom, o ambiente para desenvolvimento em C++ está preparado, mas antes de começar a escrever nossos programas, precisamos definir qual o editor de textos utilizaremos. Existem muitas excelentes opções gratuitas de editores de texto próprios para o desenvolvimento em C++, mas como na terceira fase do nosso curso vamos enfatizar o desenvolvimento utilizando o Qt, acho uma boa idéia adotar o editor do Qt Creator como editor oficial para o nosso curso.

O Qt Creator é na verdade muito mais do que apenas um editor de textos para programação em C++. É uma IDE - Integrated Development Environment (Ambiente integrado de desenvolvimento). Neste curso, no entanto, utilizaremos apenas o editor, sem aproveitar as facilidades da IDE para criação e manutenção dos projetos. Com esta abordagem, conheceremos mais sobre os projetos em Qt, pois as IDEs "escondem" muito do trabalho envolvido no processo.

É como se estivéssemos aprendendo a dirigir: usamos carros com câmbios manuais e não automáticos. Depois que sabemos dirigir podemos optar por possuir um carro com câmbio automático. De forma semelhante, depois que tiver domínio sobre o ambiente de desenvolvimento em Qt, você pode (até deve) optar por usar as facilidades de uma IDE (como o Qt Creator).

Como vamos precisar do Qt na terceira fase do curso, instale o Qt SDK, seguindo as instruções publicadas na primeira edição da Revista Qt (www.revistaqt.com).

Com o ambiente de desenvolvimento pronto e o editor de textos definido, podemos finalmente escrever o nosso primeiro programa em C++. Este programa apenas deverá imprimir na tela a mensagem: "Funcionou!". Não é muita coisa, mas como primeiro programa já serve. Vamos lá...Execute o Qt Creator (figura 1.17).

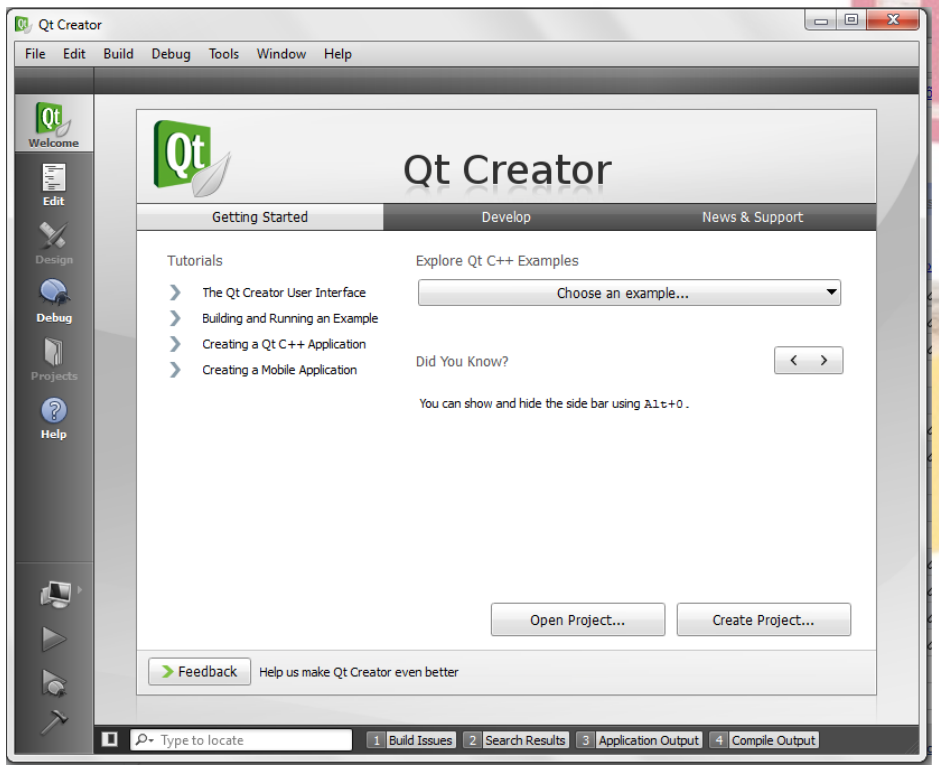


figura 1.17

Com o Qt Creator carregado, clique na opção File do Menu e selecione no menu a opção File e selecione a opção New File or Project (figura 1.18). Outra forma de fazer isso é clicando a combinação de teclas Ctrl - N.

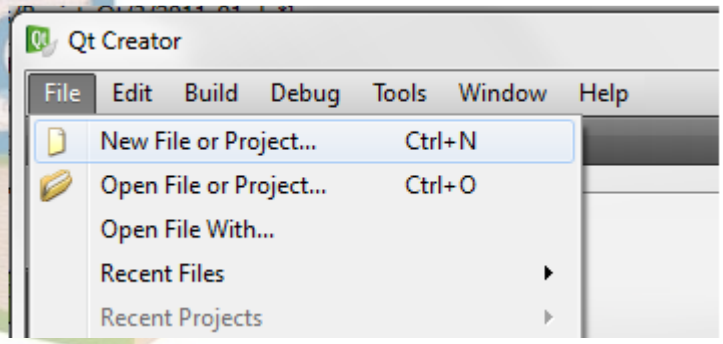


figura 1.18

Em seguida, na janela que vai ser aberta, selecione do lado esquerdo a opção C++ e do lado esquerdo a opção C++ Source File (figura 1.19). Depois de selecionar a opção "C++ Source File", clique no botão "Choose" para continuar.

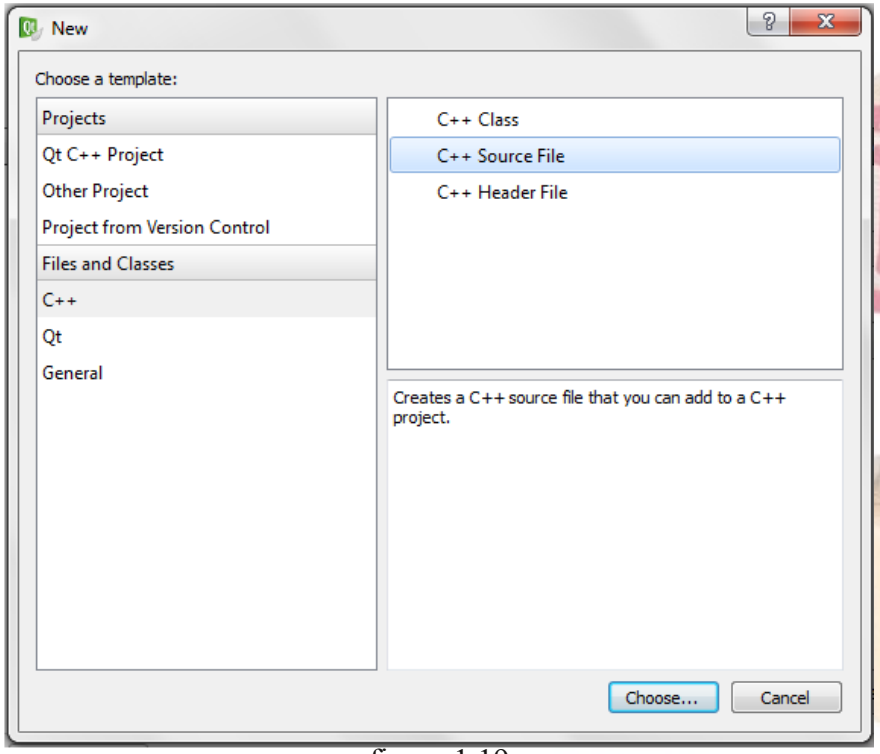


figura 1.19

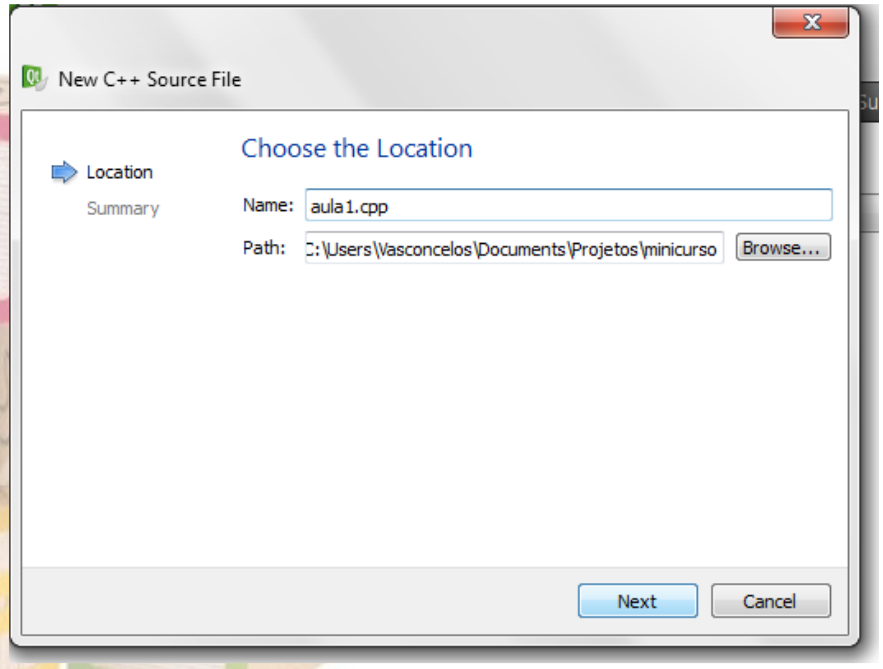


figura 1.20

Na próxma janela, você deve informar o nome e a localização do arquivo a ser criado. Para o nome, informe aula1.cpp e para a localização informe o diretório onde deseja criar o arquivo. Para facilitar referências futuras, sugiro que crie um diretório chamado minicurso e salve nele todos os arquivos criados nesta primeira fase do curso. No futuro, quando estivermos trabalhando com projetos de verdade, criaremos diretórios especíicos para cada um deles. no meu caso, criei um diretório chamado minicurso dentro de outro diretório chamado Projetos em "Meus Documentos".

No caso do Ubuntu, criei um diretório chamado minucurso dentro de um diretório chamado Projetos em meu diretório home.
Depois de informados o nome do arquivo e sua localização, clique no botão Next.

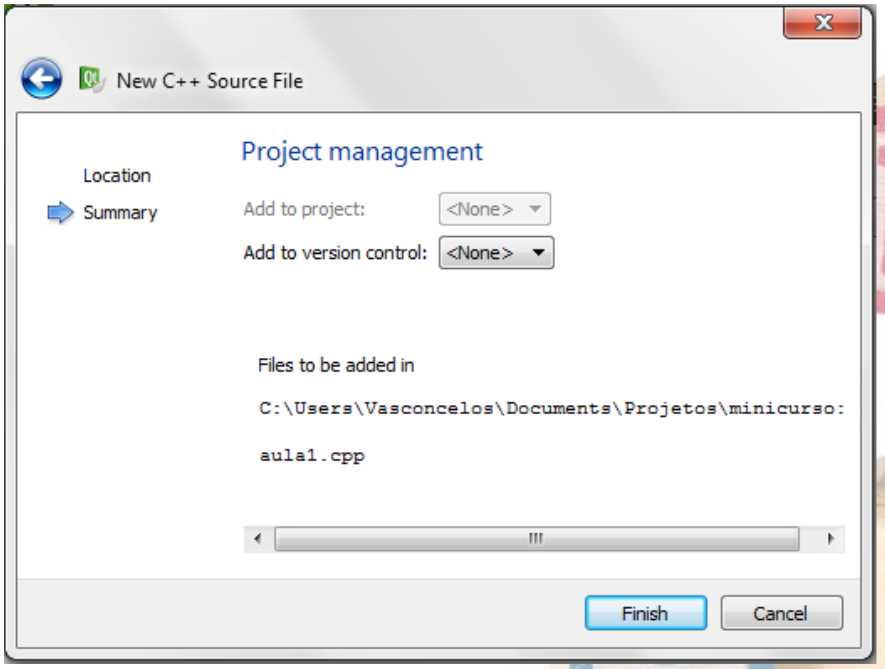


figura 1.21

Clique no botão "Finish" para que o Qt Creator prossiga com a criação do arquivo aula1.cpp.
O código do nosso primeiro programa está na figura 1.22. Por enquanto você pode não estar entendendo muito do código, mas não se preocupe. Os programas apresentados neste curso serão explicados linha a linha.

Na primeira linha do nosso programa temos a instrução:

#include <iostream>

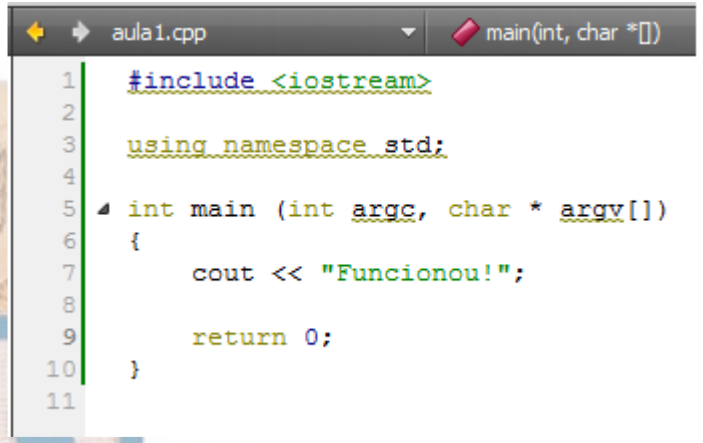


figura 1.22

Esta instrução identifica arquivos de cabeçalho a serem utilizados em nossos programas. Mais tarde veremos o que são arquivos de cabeçalho. Por ora, basta saber que esta é a forma de utilizarmos em nossos programas, códigos que estão declarados em outros arquivos. No caso do aula1, estamos usando o arquivo iostream que refere-se a uma biblioteca padrão de C++ contendo funções para entrada e saída de dados. Usaremos neste programa, um objeto definido nesta biblioteca para saída de dados (impressão na tela do monitor de vídeo).

A linha 2 foi deixada em branco para melhorar a legibilidade do programa, vejamos então a linha 3:

using namespace std;

Esta instrução serve para indicar que neste programa estamos usando o espaço de nomes (namespace) chamado std. No futuro veremos o que vem a ser um espaço de nomes (namespace).

A linha 4 também foi deixada em branco, então lá vamos nós para a linha 5:

```
int main (int argc, char  * argv[])
```

Se isto lhe parece grego, não se preocupe. Com o decorrer do curso você irá se familiarizando com a sintaxe do C/C++. Por enquanto basta saber que esta linha declara uma função chamada main. Este é o ponto de entrada dos programas em C e em C++, é a função chamada automaticamente ao executar o programa.

Na linha 6 temos um abre chaves - { - que indica o início de um bloco de código. Neste caso, o bloco que corresponde ao corpo da função main. Na linha 10 temos o fecha chaves - } - indicando o final do bloco.

Na linha 7 - primeira linha do corpo da função main, temos a seguinte instrução:

```
cout << "Funcionou!";
```

Esta instrução imprime na saída padrão - a tela do monitor de vídeo - a mensagem "Funcionou!".

A linha 8 foi deixada em branco e na linha 9 temos a instrução:

```
return 0;
```

Esta instrução indica que a função main retorna o valor inteiro 0. Em alguns casos, podemos retornar valores diferentes de 0 para indicar situações relacionadas ao encerramento do programa, como diferentes condições de erro. Veremos isto no futuro.

Salve o arquivo teclando CTRL - S e vamos compilar o nosso pequeno programa. Veremos os procedimentos de compilação no Windows e no Linux.

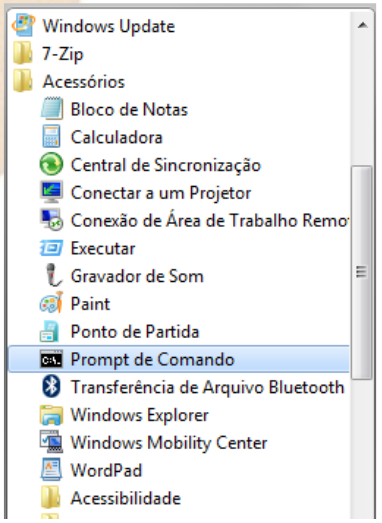


figura 1.23

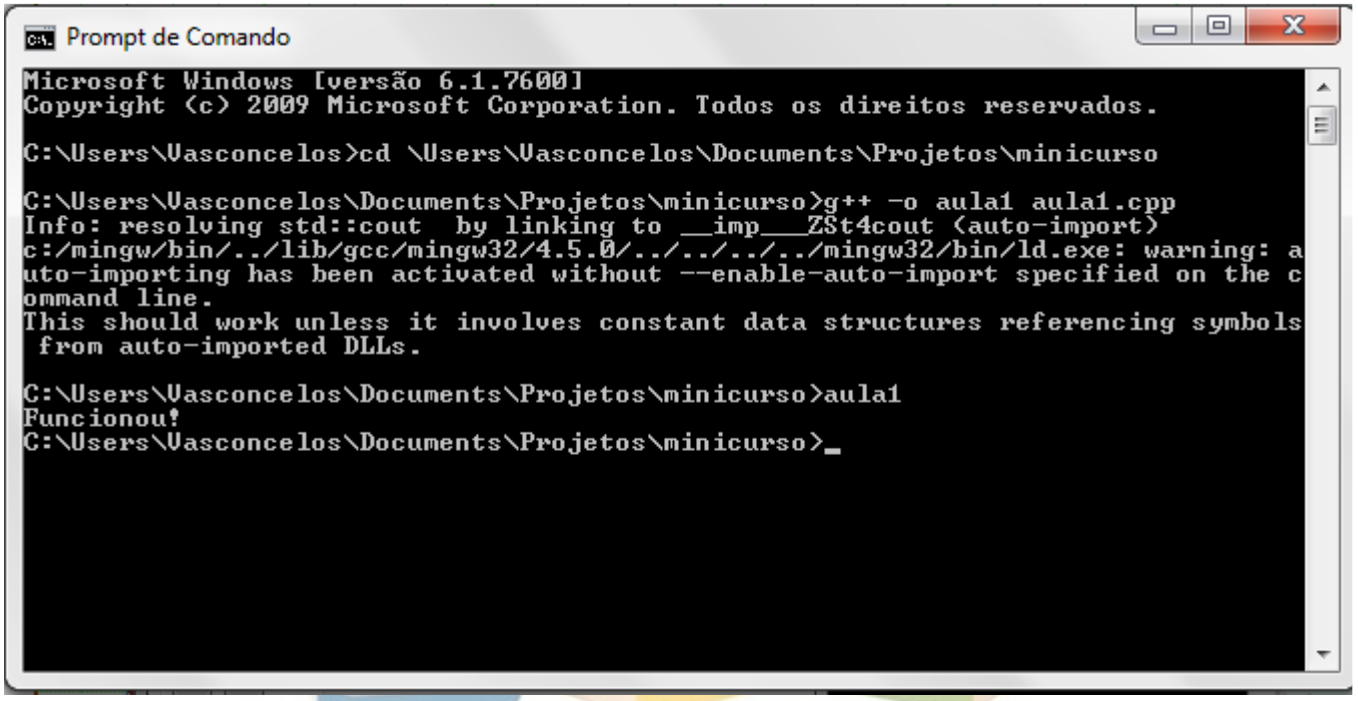


figura 1.24

Vá para o diretório onde você salvou o arquivo aula1.cpp e execute o seguinte comando:

g++ -o aula1 aula1.cpp

Este comando executa o compilador gerando um arquivo executável. O argumento "-o aula1" indica o nome do arquivo executável que será gerado. Neste caso, será gerado um arquivo chamado aula1.exe. O segundo argumento é o nome do arquivo a ser compilado - aula1.cpp.

O compilador emitiu algumas advertências, mas por ora não precisamos nos preocupar com isso.

Vejamos agora como compilar o programa na plataforma Linux.
O procedimento é bem semelhante: abra um terminal, vá para o diretório onde salvo o arquivo aula1.cpp e execute o seguinte comando:

g++ -o aula1 aula1.cpp

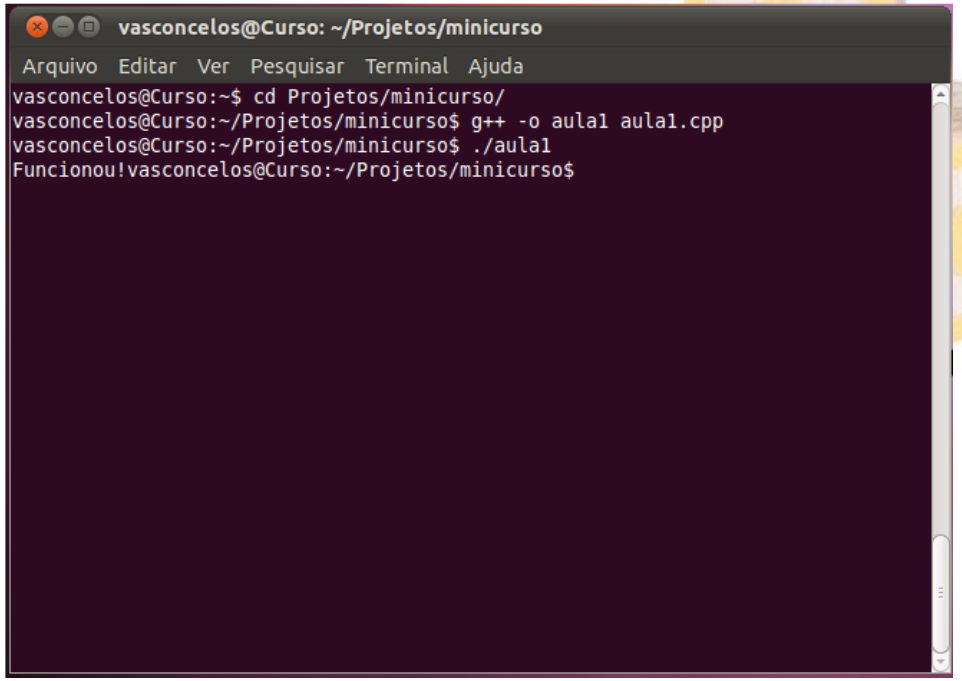


figura 1.25

O comando para compilação é o mesmo que utilizamos na plataforma Windows, mas a tivemos diferenças na execução. Para executar o programa digitamos "./aula1" ao invés de "aula1". Outra sutil diferença está no resultado da execução. No Windows a mensagem foi impressa em uma linha vazia, enquanto no Linux a mensagem ficou junto com o prompt de comando. Em breve veremos como fazer para colocar uma quebra de linha após uma mensagem impressa.

O objetivo nesta primeira aula era:

1) preparar o ambiente para desenvolvimento em C++ (Ubuntu Linux ou Windows);
2) instalar o editor de textos que utilizaremos durante o curso para escrever nossos códigos;
3) fazer um pequeno programa para confirmar que a plataforma de desenvolvimento está ok.

A partir da próxima aula começaremos a estudar a linguagem C++ propriamente dita, e teremos exercícios para colocar nossos conhecimentos à prova.

Por enquanto, fiquem à vontade para enviar seus emails para professorvasconcelos@googlegroups.com.

Até a próxima aula.

André Luiz de Oliveira Vasconcelos

Este mês foi anunciada uma parceria entra duas empresas gigantes que dispensam maiores apresentações: a finlandesa Nokia e a norte-americana Microsoft. A parceria define o Windows Phone como plataforma primária para os smartphones Nokia. Além disso, as ferramentas da Microsoft devem ser usadas para desenvolvimento de aplicações para os dispositivos Nokia com Windows Phone.

Para a "comunidade" Qt, a notícia soou como "Nokia abandona o Qt". Uma nota publicada no blog oficial do Qt (http://blog.qt.nokia.com), pelo diretor de Ecosistema Qt, Daniel Kihlberg, relaciona alguns motivos que justificariam a manutenção do Qt pela Nokia.

"Qt continuará a desempenhar um papel importante na Nokia", diz Kihlberg. Dentre os motivos, a manutenção de 200 milhões de usuários Symbian da Nokia e uma projeção da empresa para vender mais 150 milhões de dispositivos Symbian nos próximos anos.

Além disso, Kihlberg alega o crescimento do número de desenvolvedores em Qt em função da tecnologia Qt Quick e do novo SDK da Nokia/Qt, a adoção do Qt por parte de outras importantes empresas, citando inclusive a Dreamworks.

O objetivo da nota parece mais acalmar a "comunidade" Qt do que propriamente exclarecer o futuro do Qt dentro da Nokia. Há quem especule que os planos sejam extinguir o Symbian, adotando o Windows Phone como plataforma única.

Enquanto isso na Sala de Justiça...

Um desenvolvedor romeno chamado Bogdan Vatra anuncia o lançamento da versão alfa de um "port" do Qt para o sistema Android (http://br-linux.org/2011/necessitas-qt-para-android-versao-alfa/). Esta implementação de Qt para Android, chamada de Necessitas Suite é um projeto independente da Nokia e do Google.

Tratar erros de entrada do usuário costuma ser uma tarefa desgastante e que muitas vezes, ocupa grande parte do tempo de implementação de um software. Uma forma de reduzir o código para tratamento de erros é filtrar a entrada, não permitindo assim que o usuário consiga entrar com um valor inválido.
Por exemplo: Para um campo de entrada QLineEdit que só pode receber um valor numérico inteiro, temos que impedir que o usuário possa digitar qualquer caractere que não seja um número no intervalo de (0...9) ou um sinal de menos (-).

Para isso, poderíamos reimplementar o evento KeyPress para impedir a entrada de caracteres inválidos. Mas somente evitar que sejam digitados caracteres inválidos, muitas vezes não é o suficiente para evitar entradas inválidas. Por exemplo: O usuário poderia digitar 92-1 o que não corresponde a um número inteiro válido, pois o sinal de menos só poderia aparecer na primeira posição da string.

Para eliminar esse problema poderíamos fazer uma rotina de validação que é executada quando o texto está sendo editado. Quando estamos implementando um software que contem muitos campos que devem ser filtrados, o trabalho de filtragem se torna muito cansativo e o código muito grande. Em muitos casos podemos usar máscaras de entrada, mas às vezes elas são limitadas ou não aplicáveis.

Rafael Fassi Lobão

O Rafael é bacharel em Engenharia Mecatrônica e enviou este artigo sobre validação de entradas de dados como contribuição para a Revista Qt. Quem quiser segui-lo no Twitter, o endereço é: @Rafael_Fassi

Para resolver esse problema e simplificar nosso trabalho de validação da entrada, existe no Qt a classe abstrata QValidator que serve para validar entradas de texto. Desta se derivam três subclasses, que são:

- QIntValidator – Valida a entrada de números inteiros.
- QDoubleValidator – Valida a entrada de números fracionários.
- QRegExpValidator – Valida a entrada com uma expressão regular personalizada.

Se quisermos que em um determinado QLineEdit o usuário só possa digitar um valor numérico inteiro poderíamos usar o QIntValidator desta forma:

```
QIntValidator *intValidator = new QIntValidator(this);
QLineEdit *lineEdit = new QLineEdit(this);
lineEdit->setValidator(intValidator);
```

Vamos supor que estamos desenvolvendo um aplicativo de controle de vendas. Quando o vendedor selecionar um produto, uma variável chamada QuantDisponivel irá indicar quantas unidades do produto estão disponíveis no estoque.

Queremos que o vendedor entre com a quantidade do produto a ser vendida em um QLineEdit. Mas precisamos evitar que ele digite algum valor que não seja um número inteiro. Também queremos evitar que o vendedor digite um valor acima da quantidade disponível em estoque. Isso poderia ser implementado setando o range de QIntValidator desta forma:

QIntValidator *intValidator = new QIntValidator(1, QuantDisponivel, this);

Assim, estamos passando o range logo no construtor. Podemos também modificar o range utilizando as funções setRange, setButton ou setTop.

Um recurso fantástico para a validação de entrada é o QRegExpValidator. Não vou entrar em detalhes sobre expressões regulares, mas um bom resumo pode ser encontrado em:

http://guia-er.sourceforge.net/index.html

Para demonstrar o QRegExpValidator, vamos supor que em um campo QLineEdit do nosso aplicativo, precisamos que o usuário digite um valor hexadecimal de 0 a FFFF. Para isso, podemos usar o QRegExpValidator assim:

```
QValidator *hexValidator = new QRegExpValidator(QRegExp("[0-9A-Fa-f]{1,4}"), this);
```

Desta forma o usuário só poderá digitar números no intervalo de (0...9), letras minúsculas (a...f) e maiúsculas (A...F). Para a expressão ser válida ela deverá ter no mínimo um dígito e só poderão ser digitados no máximo 4 dígitos.
Com a expressão regular, podem ser feitos os mais diversos tipos de validadores de entrada personalizados. Como outro exemplo de expressão regular, esta serve para entrada de horas no formato (hh:mm) "([01][0-9]|2[0-3]):[0-5][0-9]".
Nossa entrada vai até 23h e 59m. Dessa forma se o primeiro dígito da hora for 0 ou 1, o segundo pode ir até nove, o que poderia ser (09h) ou (19h). Mas se o primeiro dígito for 2, o segundo só pode ir até 3 que seria o nosso máximo para hora (23h). Neste caso se utiliza o OU, que é representado pelo símbolo (|). Então, a hora pode ser:

Com o primeiro digito (0 ou 1) e o segundo de (0...9)

OU (|)

Com o primeiro dígito igual a dois e o segundo de (0...3)

Para os minutos, como vão até 59, ficaria assim:
O primeiro pode ser de (0...5) e o segundo de (0...9)

Além de usarmos as subclasses QIntValidator, QDoubleValidator e QRegExpValidator, também podemos herdar a classe virtual QValidator e implementar nossa própria classe de validação. Uma vantagem disso é que podemos não somente validar a entrada como também fazer transformações no texto de entrada. Por exemplo, podemos validar a entrada e transformá-la em maiúscula.

Para demonstrar, vamos criar um validador de entrada hexadecimal que pode permitir entrada negativa (-FF) e pode transformar a entrada para maiúscula.

Aqui está nosso arquivo de cabeçalho:

```
#ifndef FVALIDATORS_H
#define FVALIDATORS_H
#include <QValidator>
class FHexValidator : public QValidator
{
    Q_OBJECT
public:
    explicit FHexValidator(QObject *parent = 0, bool _Signal = false, bool
_UpperCase = true);
    void setSignal(bool _Signal) { Signal = _Signal; }
    void setUpperCase(bool _UpperCase) { UpperCase = _UpperCase; }
    QValidator::State validate(QString &input, int &pos) const;
private:
    QRegExpValidator *validator;
    bool Signal;
    bool UpperCase;
};
#endif // FVALIDATORS_H
```

Criamos uma classe FHexValidator que é herdada de QValidator e reimplementamos a função virtual pura validate.
Estou usando QRegExpValidator para simplificar o trabalho de filtrar a entrada hexadecimal, mas uma rotina que faça isso poderia ser implementada facilmente.
A variável Signal indica se a entrada pode ser negativa e a variável UpperCase indica que a entrada deve ser convertida em maiúscula.

Vamos ao código:

```
#include "fvalidators.h"

FHexValidator::FHexValidator(QObject *parent, bool _Signal, bool _UpperCase):
        QValidator(parent)
{
    validator = new QRegExpValidator(QRegExp("[-0-9A-Fa-f]{1,8}"), this);
    Signal = _Signal;
    UpperCase = _UpperCase;
}
QValidator::State FHexValidator::validate(QString &input, int &pos) const
{
    if(input.contains('-'))
    {
        if(!Signal || input[0] != '-' || input.count('-') > 1) return Invalid;
      if(input.size() < 2) return Intermediate;

    }
    if(UpperCase) input = input.toUpper();
    return validator->validate(input, pos);
}
```

No construtor iniciamos validator para validar uma entrada hexadecimal com até 8 dígitos que pode conter o sinal de (-).

A implementação de validate é bem simples. Ele pode retornar 3 estados, que são:

Invalid – Quando a entrada é inválida.
Intermediate – Quando ainda não é determinado se a entrada é válida.
Acceptable – A entrada é válida.

Primeiro a função checa se a string de entrada contém o caractere (-). Caso contenha a entrada pode ser inválida se o modo de entrada negativa (Signal) não estiver habilitado ou se o caractere (-) não estiver na primeira posição da string. Somente com essas condições poderia acontecer um erro se o usuário digitar (-), voltar o cursor para a primeira posição e digitar novamente (-). Para isso não acontecer a entrada também será invalidada se a string contiver mais de um caractere (-).
Se a string contém o sinal (-) e não foi invalidada, precisamos saber se existem mais dígitos, pois uma string que contém somente o sinal (-) não é aceitável, mas sim indeterminada. Pois necessita de algum dígito para ser válida.

Logo após, se o modo UpperCase estiver ativo, o texto é transformado em maiúsculo.

Finalmente a string pode ser validada pelo objeto validator, que vai verificar se nela existem somente dígitos hexadecimais e o sinal (-).

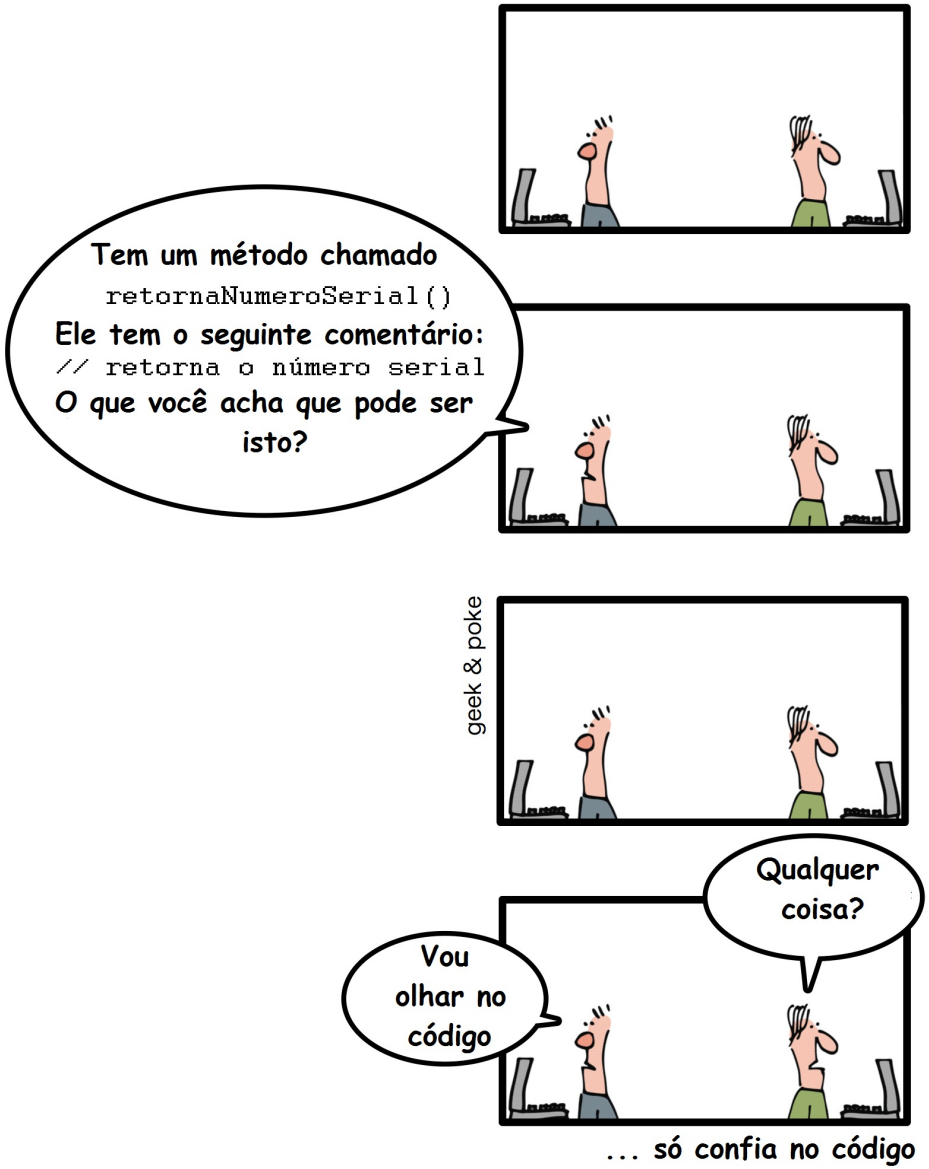
É isso aí pessoal, vou ficando por aqui e espero que tenham gostado do artigo.
Até a próxima!

Oliver Widder é o criador do site "Geek and Poke" e colaborador da Revista Qt.

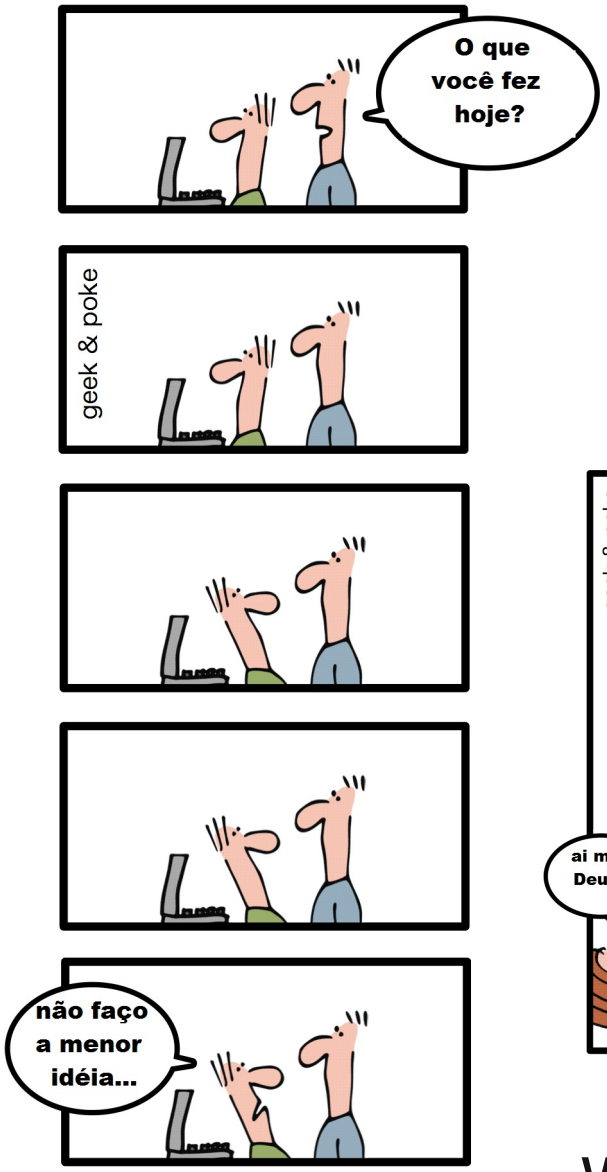
Do que precisamos

Parte 1: Um Google para as abas abertas

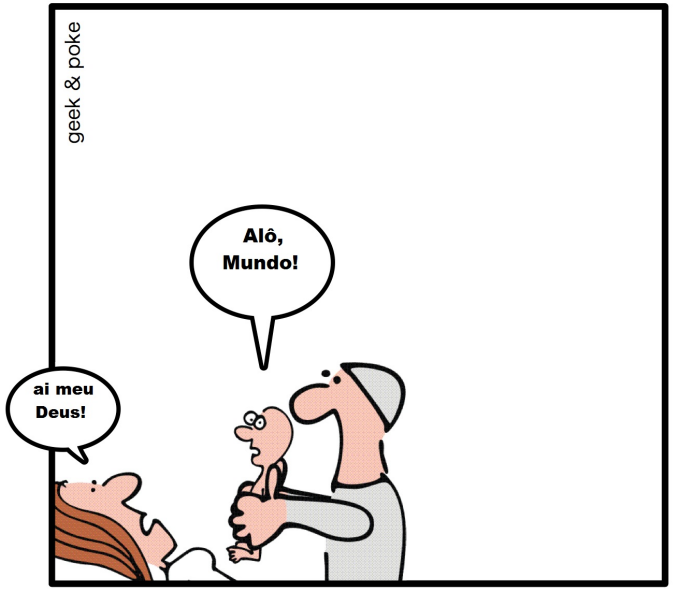
O programador de verdade...



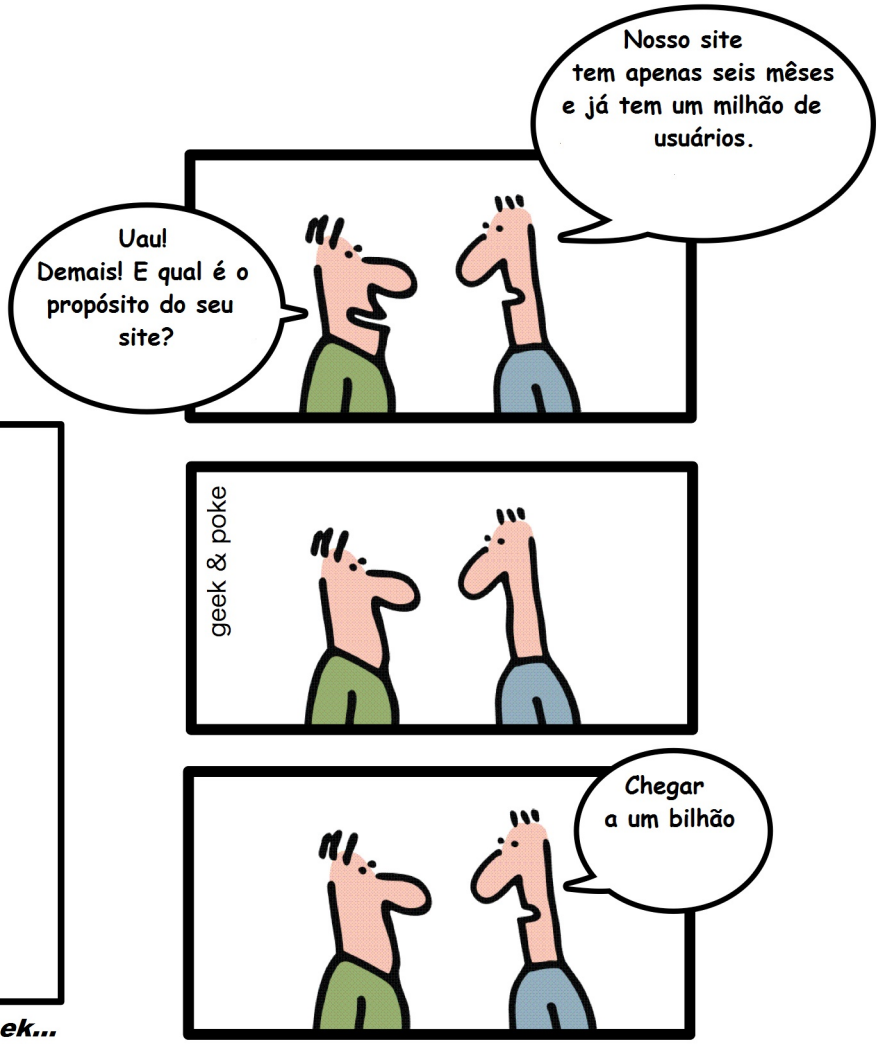
... só confia no código



Final da tarde de um programador



Nasce um Geek...



www.geekandpoke.com

InstallJammer

Na segunda edição da Revista Qt, saiu um artigo sobre a distribuição de aplicações feitas em Qt. O presente artigo pode ser considerado como uma continuação daquele. Desta vez veremos como criar um instalador de verdade para nossas aplicações, usando o InstallJammer.
O instalador simplifica o processo de instalação de nossas aplicações pois não exigem dos usuários comuns conhecimentos sobre programas de compactação e coisas do gênero. Normalmente o usuário precisa apenas responder a algumas poucas perguntas para proceder a instalação da maioria dos programas.
Existem alguns programas gratuitos para gerar instaladores. A escolha pelo InstallJammer foi feita porque além de grátis ele é multiplataforma.
Pra começar, faça o download do InstallJammer pelo endereço:

www.installjammer.com/download

A instação do InstallJammer é muito simples, não sendo necessário um tutorial para isso.

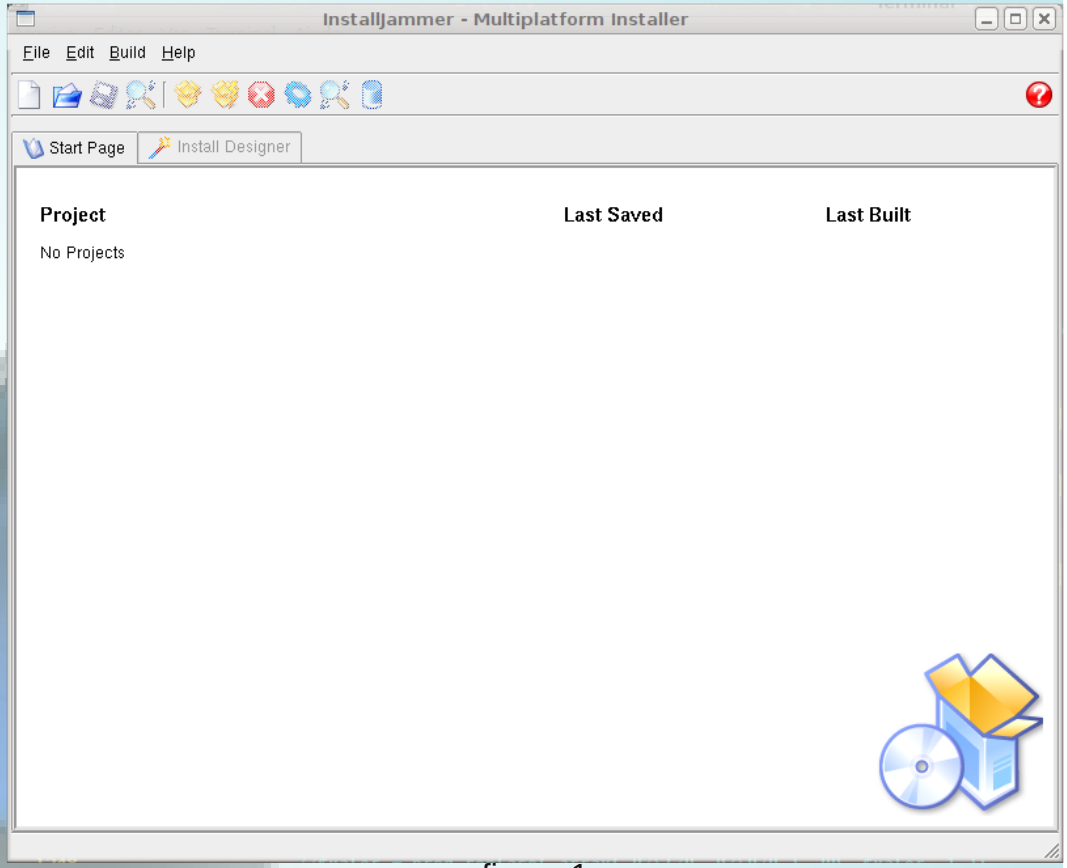


figura 1

Vou demonstrar a criação de um pacote de instalação para uma aplicação chamada Agenda (publicada nesta edição da Revista Qt).

A figura 1 mostra a tela do InstallJammer. Para criar um novo projeto, selecione a opção File -> New Project Wizard no menu do InsallJammer.

No primeiro passo da montagem do instalador da nossa aplicação, devemos informar:

- O Nome do projeto - Neste caso, Agenda.

- O Diretório raiz do projeto. O diretório sugerido pelo InstallJammer é InstallJammerProjects no diretório home do usuário. No caso do Windows, o diretório sugerido será installJammerProjects na pasta "Meus Documentos" do usuário.

Será criado pelo InstallJammer um diretório com o nome do projeto a partir do diretório raiz do projeto. No caso deste exemplo, o diretório da aplicação ficou como "/home/vasconcelos/InstallJammer/Agenda".
Para continuar, clique no botão Next para especificar outras opções do instalador (figura 2).

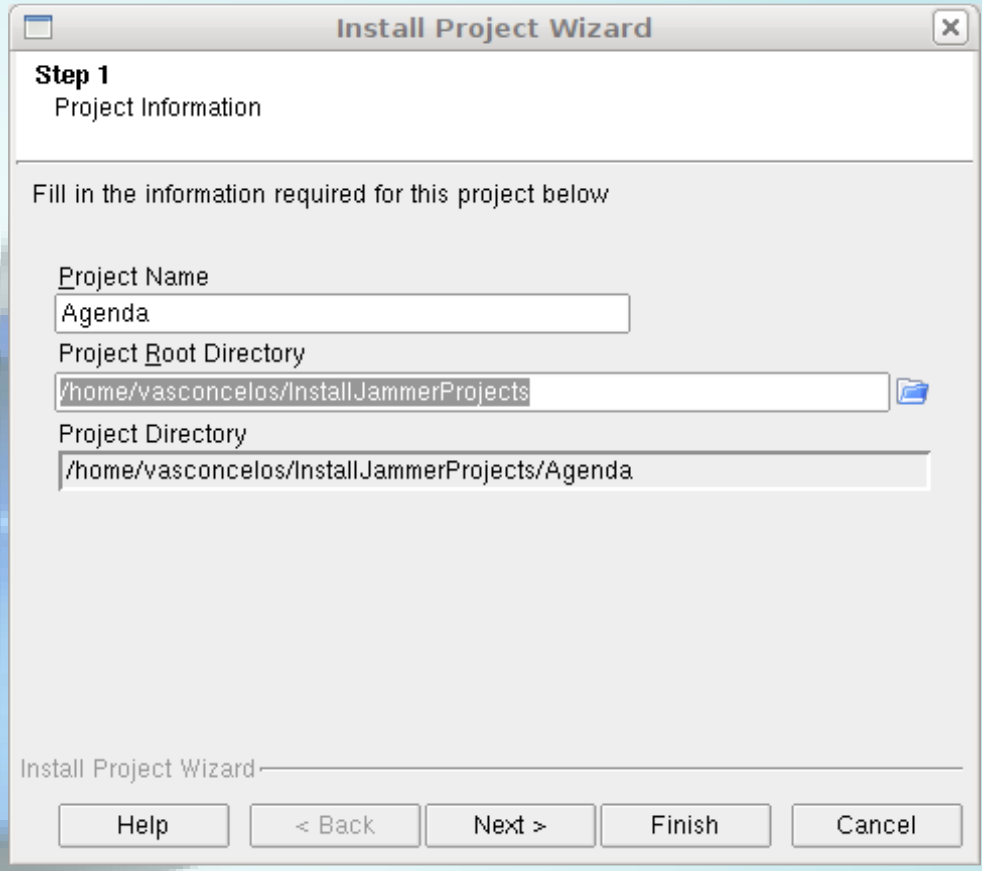


figura 2

Na próxima tela (figura 3), informe

- O nome da aplicação
- O nome curto da aplicação
- A versão da aplicação
- O nome da empresa

Em seguida clique no botão Next para continuar.
Na próxima tela (figura 4) informamos a versão do instalador (na tela anterior informamos a versão da aplicação) e os nomes dos executáveis para Windows e Unix (ou Linux).

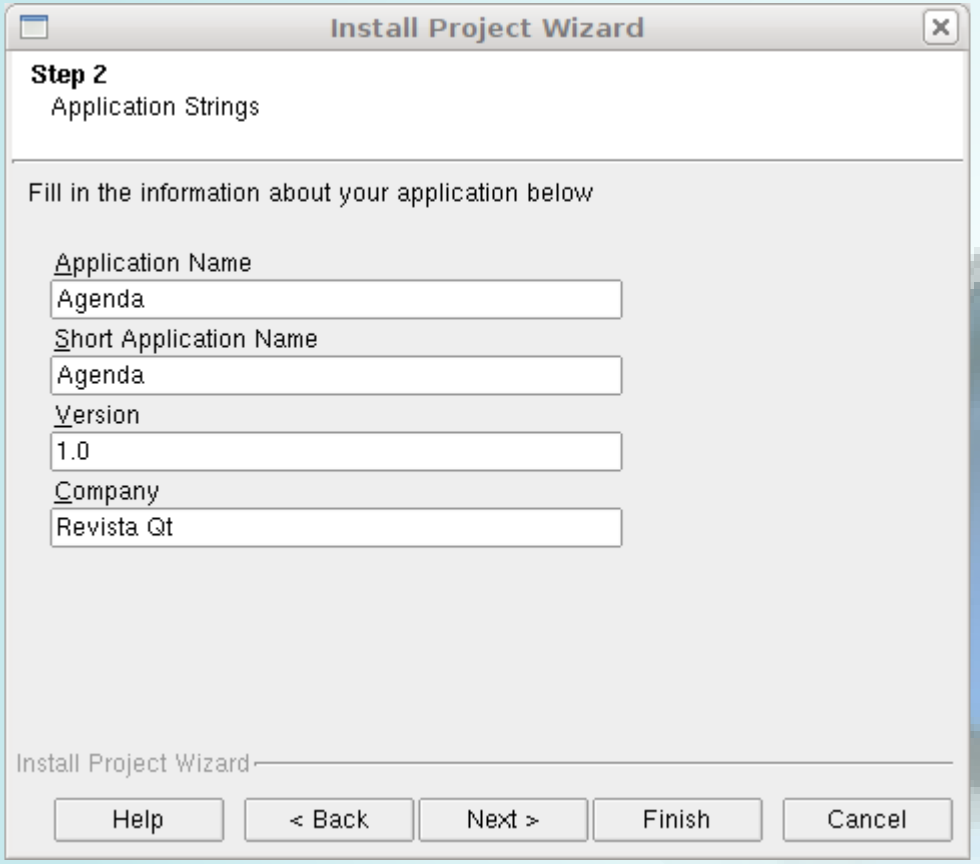


figura 3

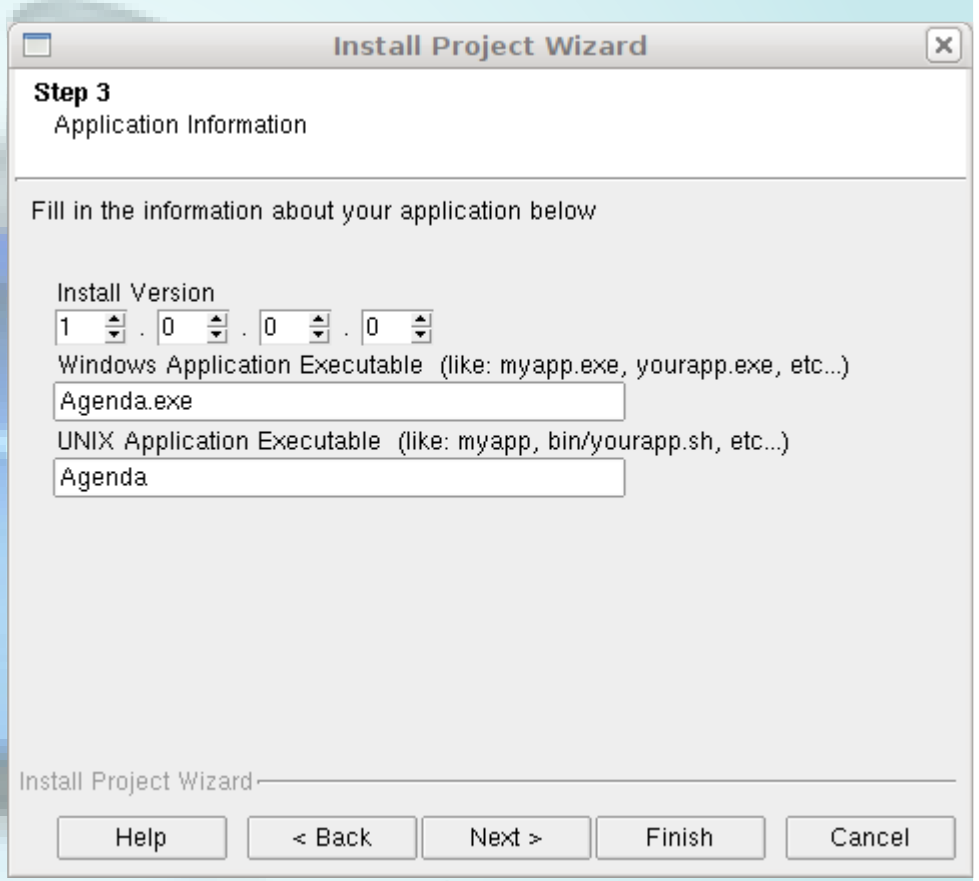


figura 4

O próximo passo é informar o diretório onde está a sua aplicação.

Observe pela figura 5 que o diretório no meu caso é o

"Agenda-build-desktop"

Este é o padrão da versão 2.01 do Qt Creator que foi a que utilizei para compilar a aplicação Agenda - o nome da aplicação acrescido de "-build-desktop. Este diretório corresponde àquele em que se encontra o executável e demais arquivos que irão compor o pacote de instalação. Se você estiver em dúvida sobre quais arquivos devem ser incluídos para a montagem do pacote de instalação da sua aplicação, consulte o artigo "Distribuindo suas aplicações Qt para Linux" na segunda edição da Revista Qt.
Para continuar, clique novamente no botão Next.
Na próxima tela (figura 6) temos a seleção do tema de instalador, determinando que aparência ele terá. No meu caso selecionei o Modern Wizard. Clique no botão Next para continuar.

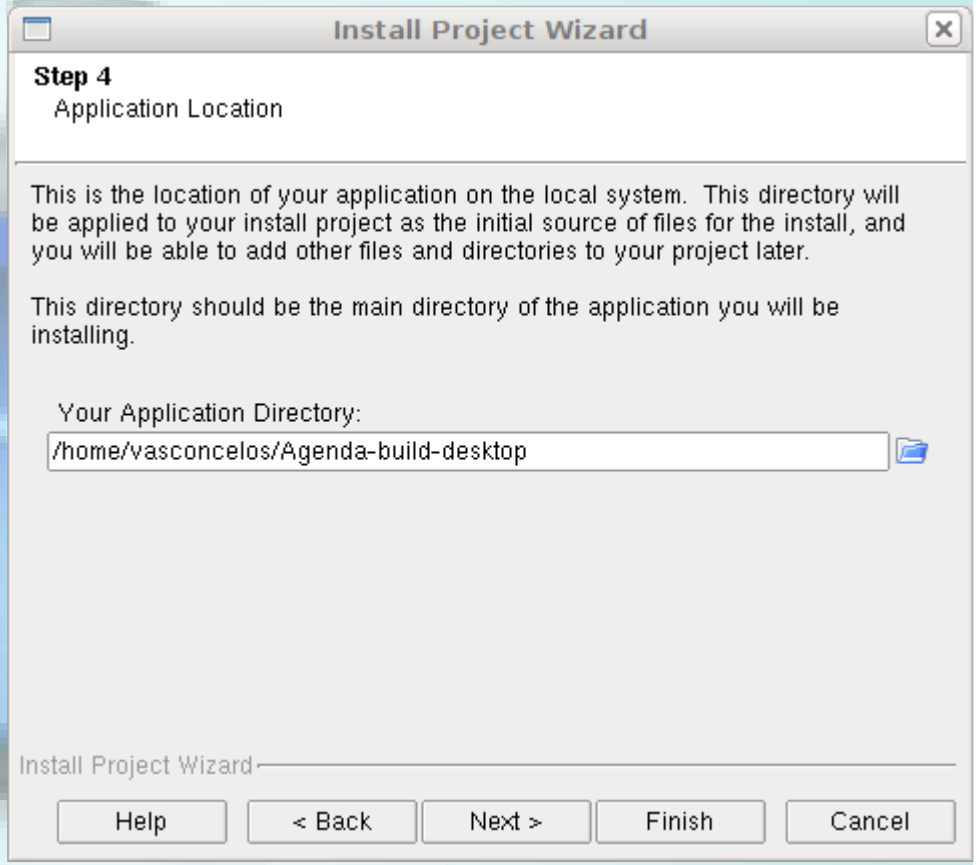


figura 5

No sexto passo da criação do instalador (figura 7), selecione a plataforma para a qual está gerando o pacote de instalação. No caso do exemplo, estou gerando um pacote para instalação na plataforma Linux 32 bits. Mais uma vez, clique no Next para continuar.

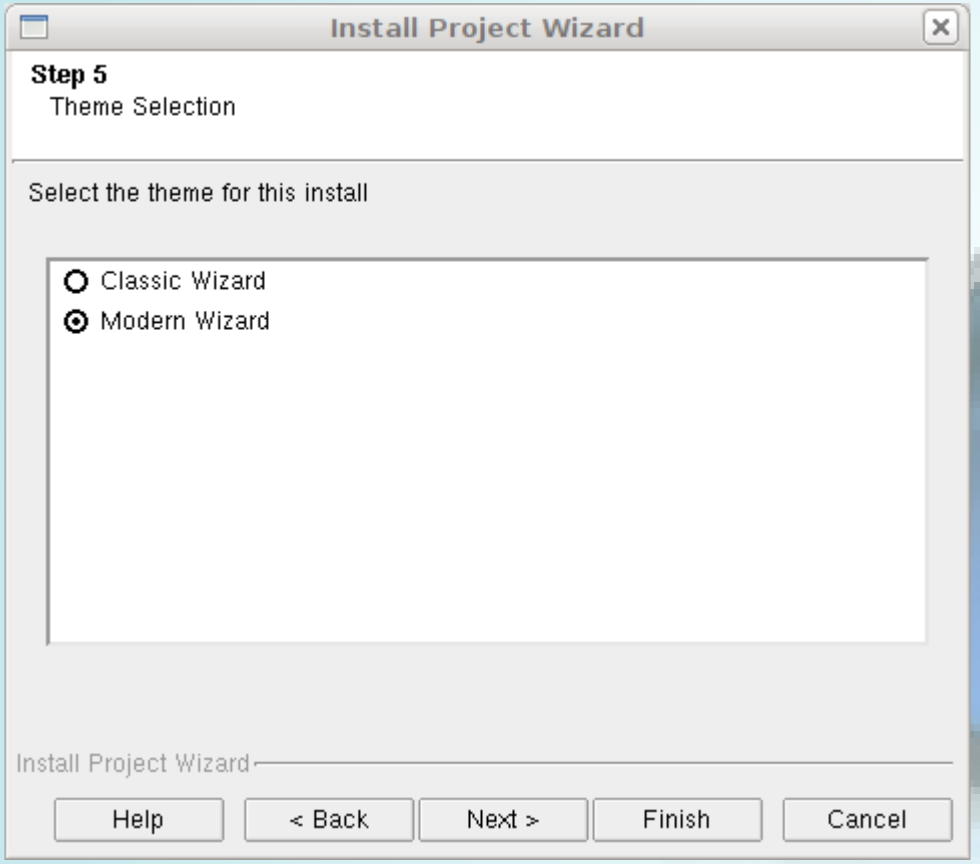


figura 6

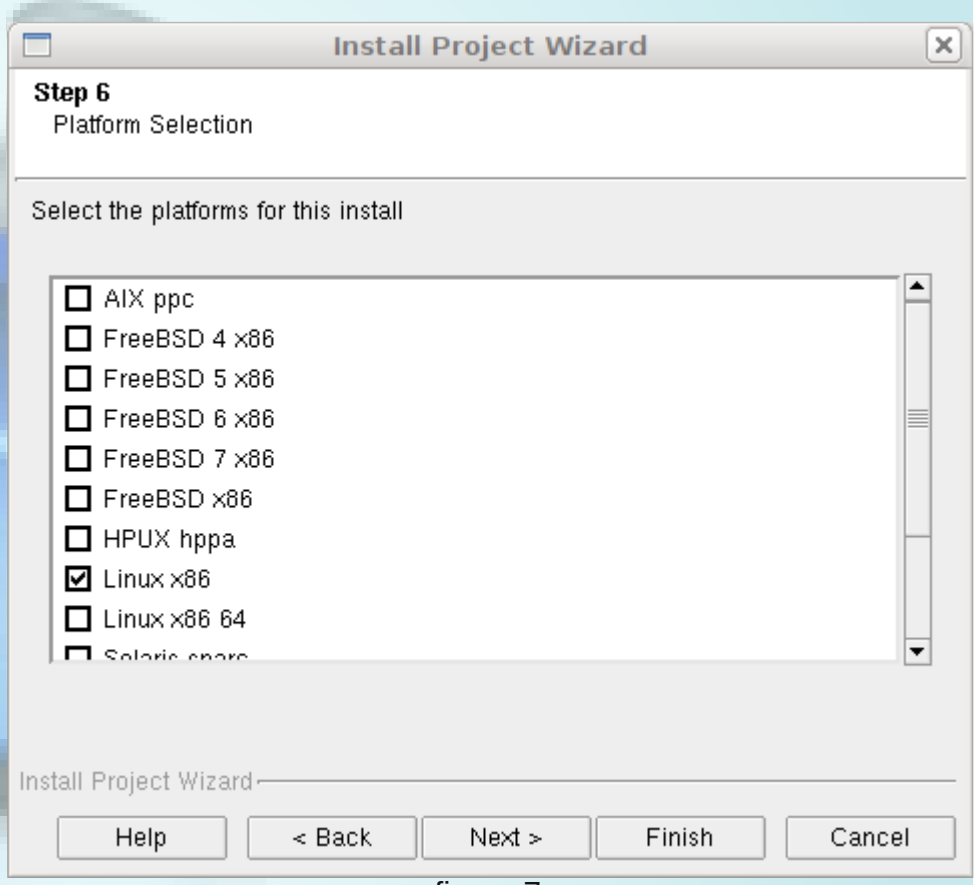


figura 7

O oitavo passo da criação do instalador permite que você selecione características adicionais ao seu instalador. Por default, todas as opções vêm marcadas. As opções disponíveis são:

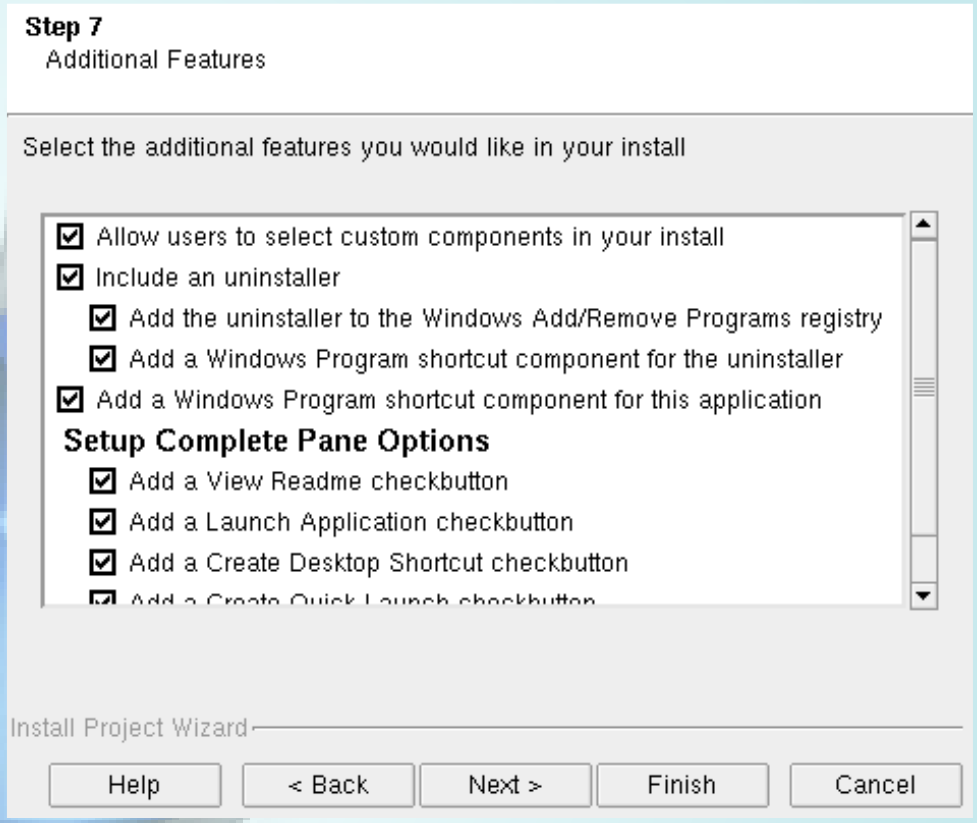


figura 8

- Allow users to select custom components in your install - Esta opção permite que o usuário selecione uma instalação personalizada, selecionando quais componentes do pacote deseja instalar.

- Include an uninstaller - Com esta opção marcada, o instalador de sua aplicação incluirá um desinstalador. Detalhando o comportamento do desinstalador, temos as opções:

- Add the uninstaller to the Windows Add/Remove Program Registry - se esta opção estiver marcada o usuário será capaz de remover a aplicação pelo Painel de Controle do Windows.

- Add a Windows Program shortcut component for the uninstaller - se esta opção estiver marcada será criado pelo Windows um atalho para o desinstalador da aplicação.

- Add a View Readme checkbutton - adiciona ao instalador um check box com a opção para abrir o arquivo readme (leia-me).

- Add a Launch Application checkbutton - inclui um check box para executar a aplicação ao final da instalação.

- Add a Create Desktop Shortcut checkbutton - inclui check box com a opção de criar um atalho para a aplicação no Desktop.

- Add a Create Quick Launch checkbutton - inclui um check box com a opção para criar um atalho para a aplicação na barra de lançamento rápido do sistema operacional.

Clique no botão Next para continuar.

Chegamos ao passo final para criação do instalador. Nesta tela temos apenas o aviso de que o InstallJammer está pronto para montar instalador. Clique no botão Finish.

Na tela mostrada na figura 10 temos um resumo com as opções do instalador que está sendo criado. Clique em Components and Files para expandir suas opções, e em seguida clique em Groups and Files. Serão mostrados os arquivos que estão no diretório da aplicação. Por default, todos os arquivos encontram-se selecionados. Desmarque os arquivos que não deseja que façam parte de sua aplicação. No caso do nosso exemplo, desmarquei os arquivos com a extensões .o, .cpp e .h, além do Makefile (figuras 11 e 12). Como a minha aplicação será executada pelo script start (veja o artigo "Distribuindo suas aplicações Qt para Linux" na segunda edição da Revista Qt), alterei a opção "Program Executable" que fica em "General Information -> Platform Information". Substitui <%InstallDir%>/Agenda por <%InstallDir%>/start (veja as figuras 13 e 14).

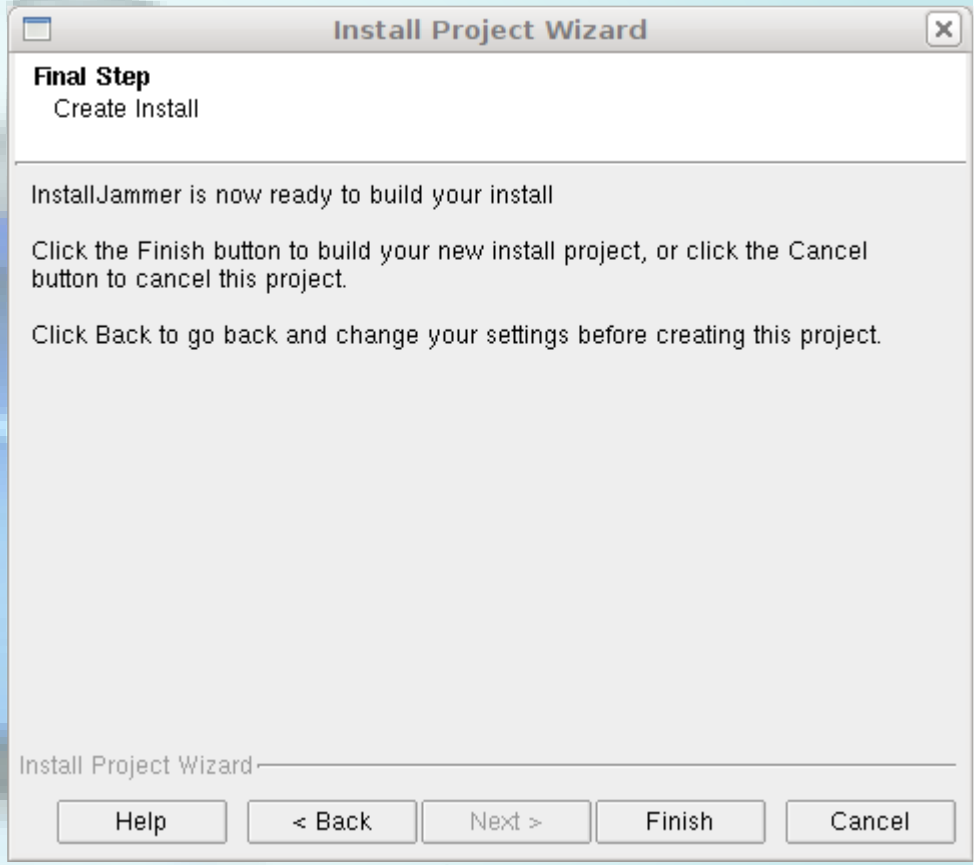


figura 9

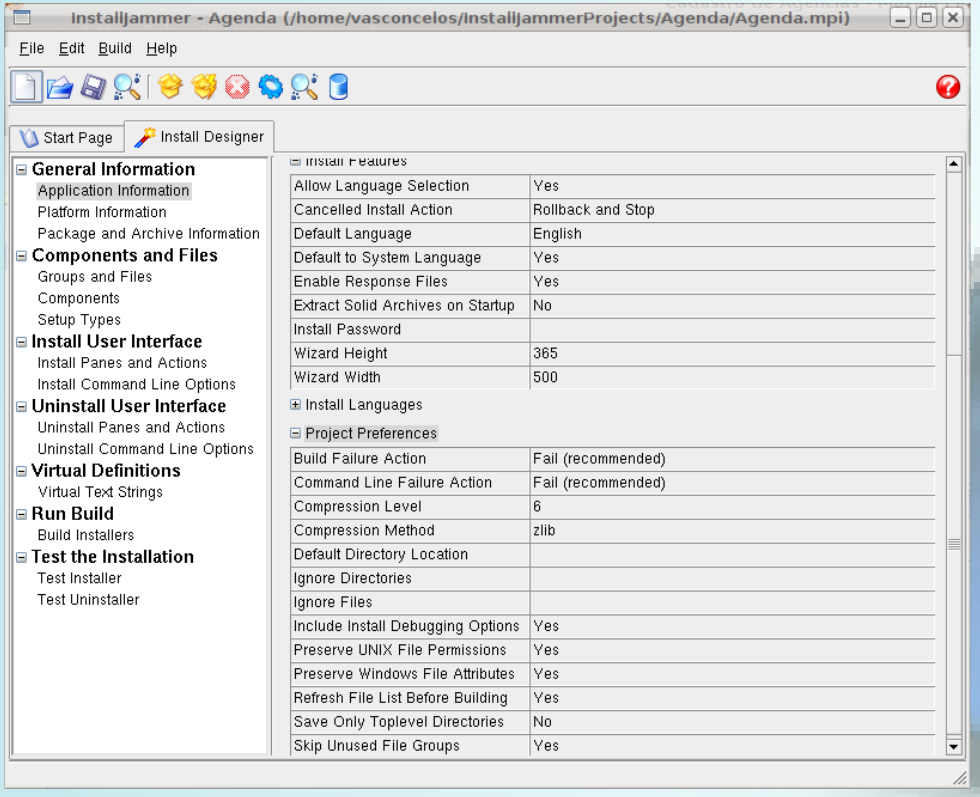


figura 10

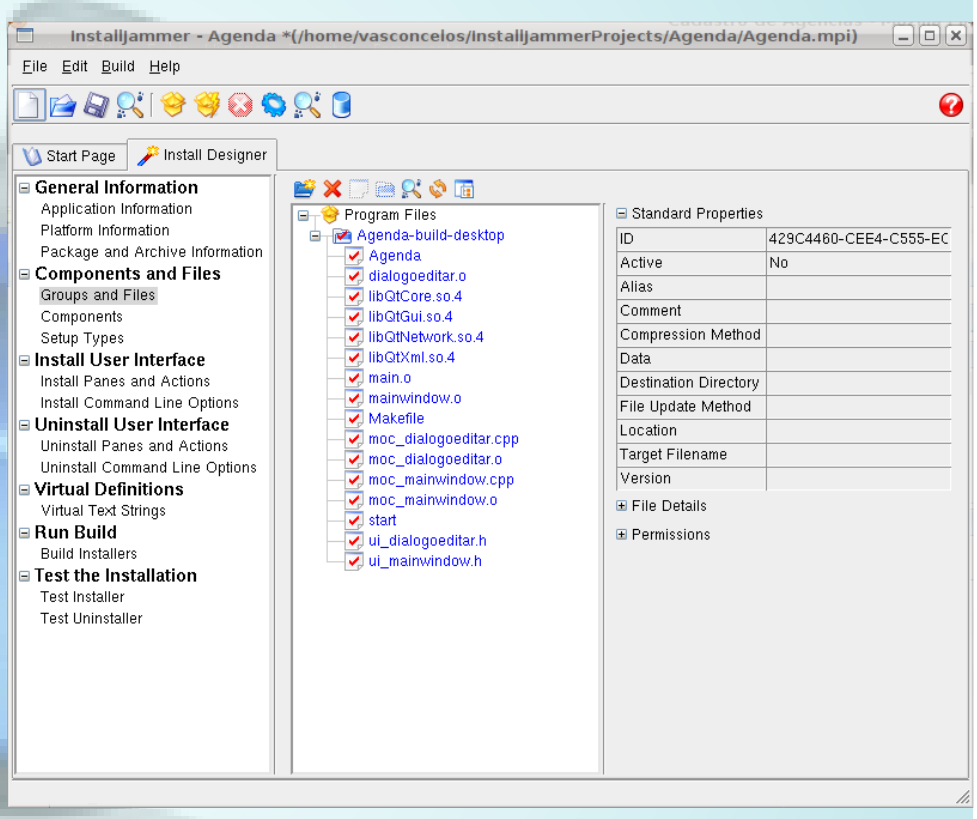


figura 11

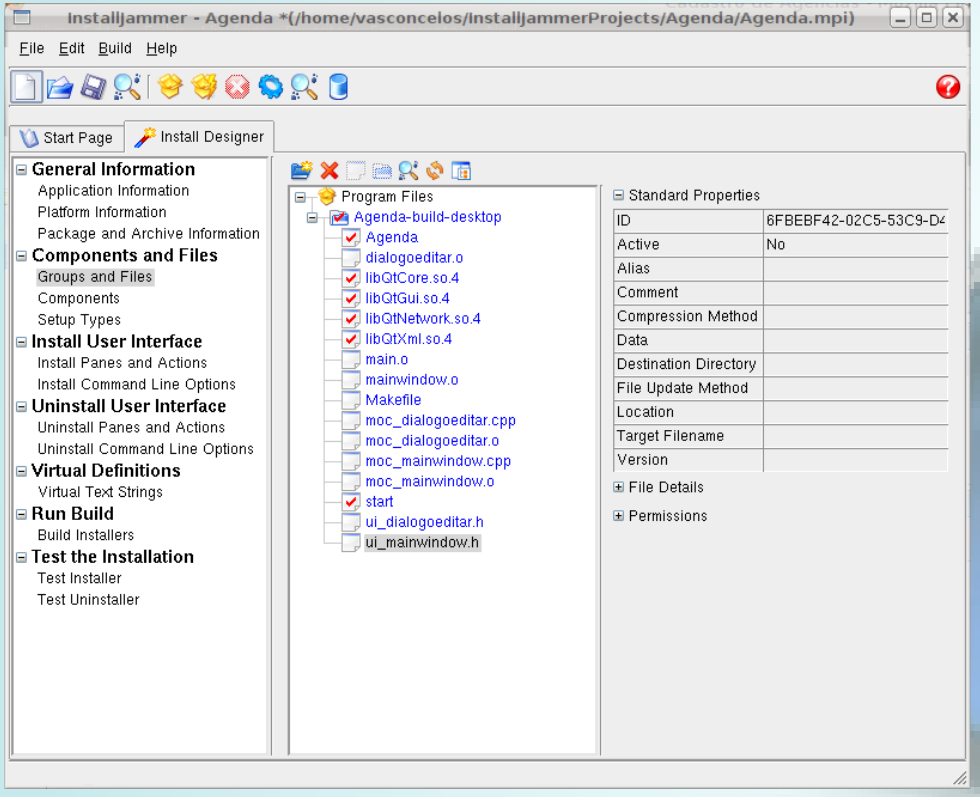


figura 12

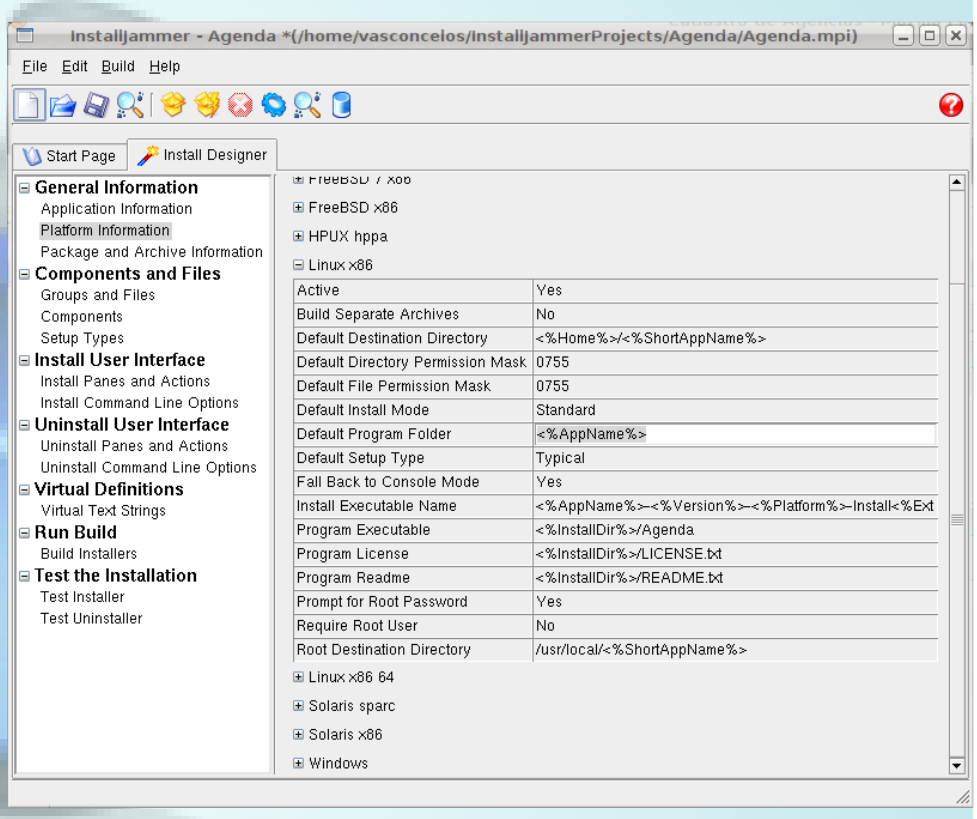


figura 13

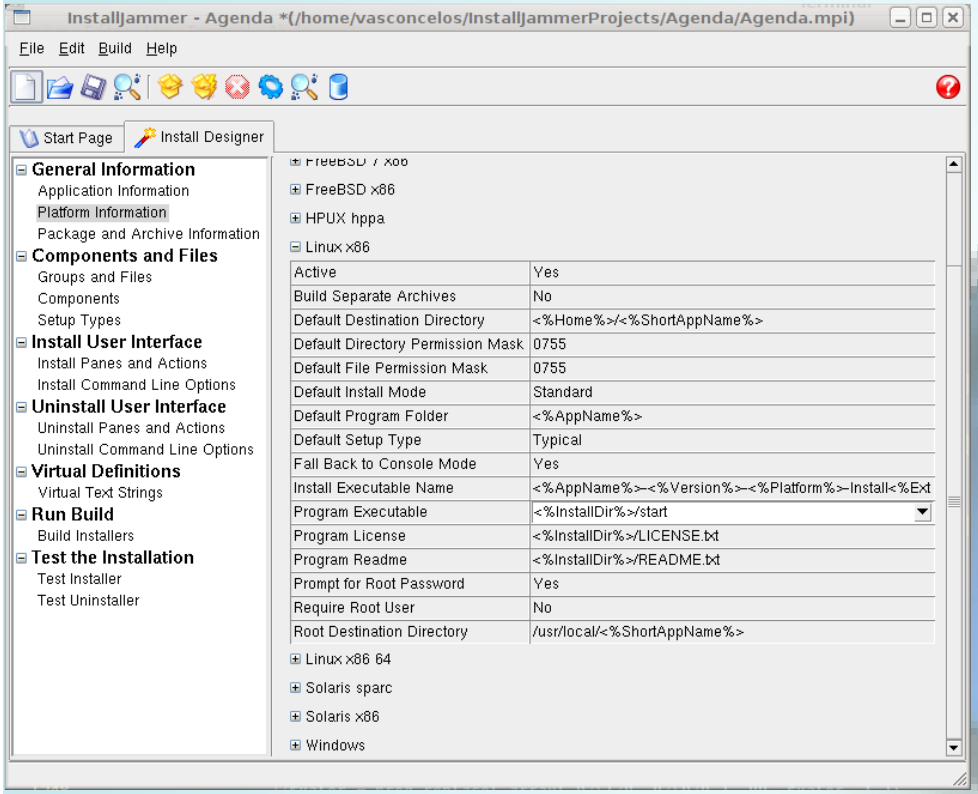


figura 14

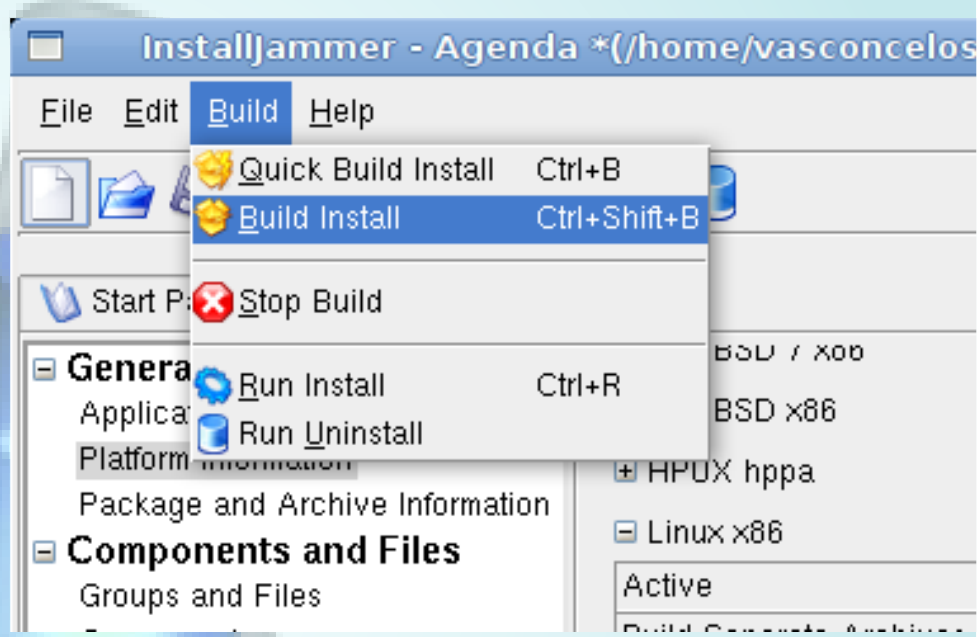


figura 15

Neste ponto podemos montar o nosso instalador. Para isso, selecione a opção Build -> Build Install do menu do InstallJammer (figura 15).
O Install Jammer vai criar um executável para instalação da sua aplicação, que pode ser então distribuido para seus usuários.

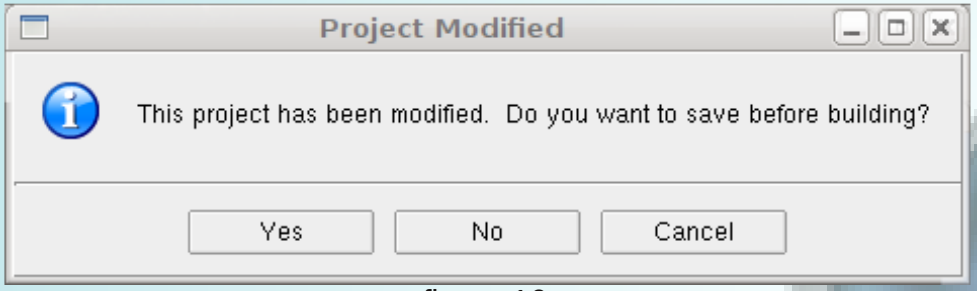


figura 16

Vimos neste artigo apenas as opções básicas do InstallJammer necessárias para a criação de um instalador.

Cada tela do instalador pode ser customizada, mas para apresentar todas as opções disponíveis precisaríamos de uma edição exclusiva da Revista. Sugiro que você explore as opções do InstallJammer verificando como afetam o instalador de sua aplicação.

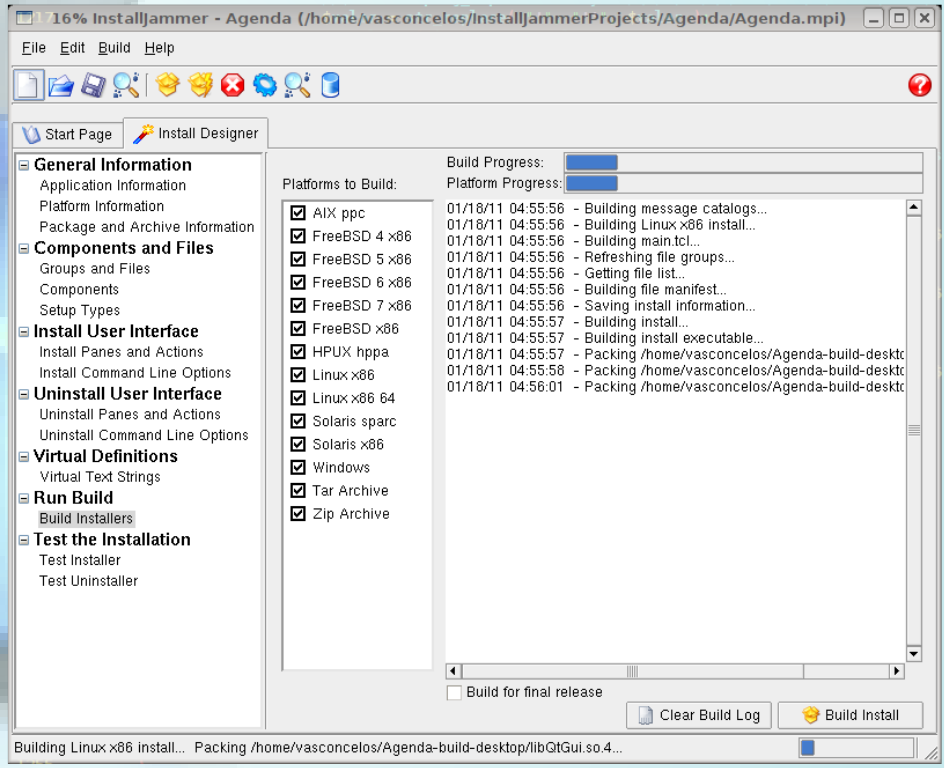
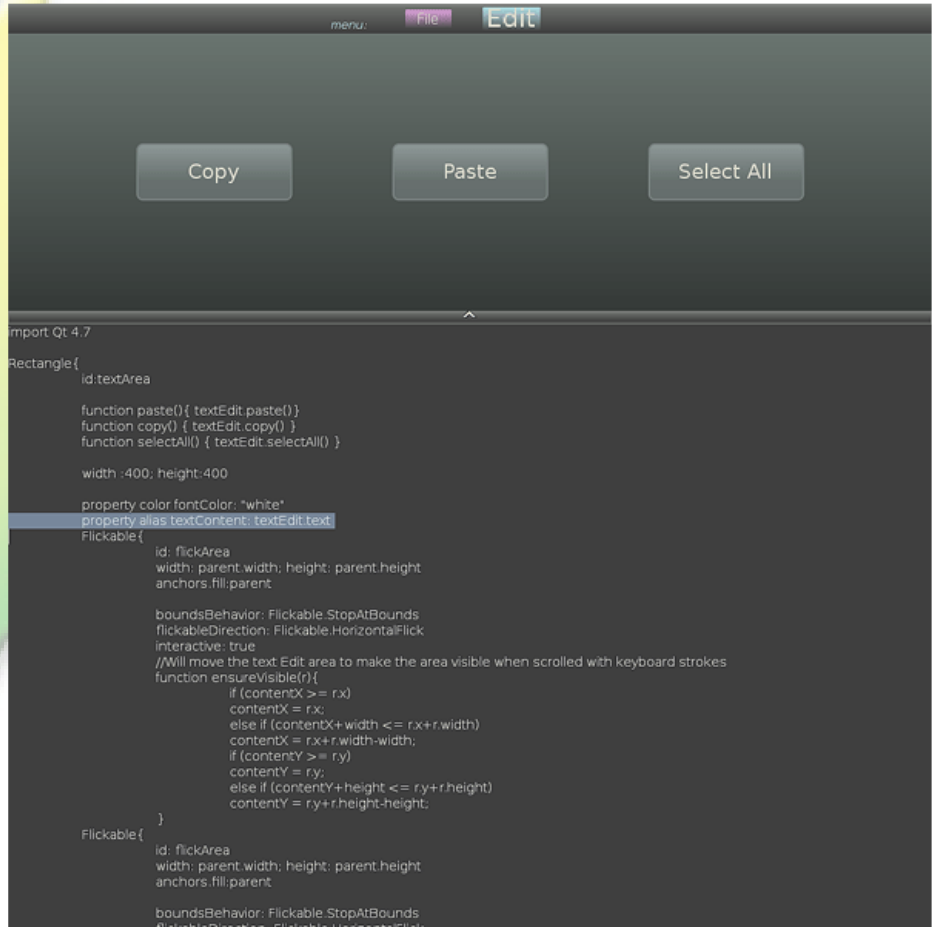


figura 17

Iniciando na Programação com QML

Como de costume, trago nesta edição da Revista Qt mais uma tradução de um documento disponível no site de documentação do Qt. Desta vez é uma versão do tutorial "Getting Started Programming with QML", cuja versão original pode ser vista pelo endereço: http://doc.qt.nokia.com/4.7/gettingstartedqml.html



Iniciando na Programação com QML

Bem vindo ao mundo do QML, a linguagem declarativa de Interfaces de Usuários (UI - User Interfaces). Neste guia de iniciação, vamos criar um aplicação simples de editor de textos usando QML. Depois de ler este guia, você deverá estar pronto para desenvolver suas próprias aplicações usando QML e Qt com C++.

QML para montar interfaces

A aplicação que construiremos é um simples editor de textos que irá carregar, salvar e desempenhar algumas manipulações de textos. Este guia consistirá de duas partes. A primeira envolverá desenhar o layout da aplicação e seus

comportamentos usando linguagem declarativa em QML. Para a segunta parte, a carga e salvamento do arquivo serão implementados usando Qt com C++.

Usando o Sistema de Meta-Objetos do Qt, podemos expor funções C++ como propriedades que elementos QML pode usar. Utilizando QML e Qt C++, podemos eficientemente desacoplar a lógica da interface da lógica da aplicação.

Para executar o código QML do exemplo, simplesmente execute a ferramenta qmlviewer com o arquivo QML como argumento. A porção C++ deste tutorial assume que o leitor possua conhecimentos básicos dos procedimentos de compilação do Qt.

Capítulos do tutorial:

1. Definindo um Botão e um Menu
2. Implementando uma Barra de Menu
3. Construindo um Editor de Textos
4. Decorando o Editor de Textos
5. Extendendo QML usando Qt C++

Definindo um Botão e um Menu

Componentes Básicos - um Botão

Começamos nosso editor de textos pela montagem de um botão. Funcionalmente, um botão tem uma área sensível ao mouse e um rótulo (label). Botões executam ações quando pressionados pelo usuário.

Em QML, o item visual básico é um elemento Rectangle. O elemento Rectangle tem propriedades para controlar a aparência e localização do elemento.

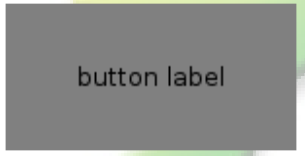
Primeiro, a declaração "import QtQuick 1.0" permite à ferramenta qmlviewer importar os elementos QML que usaremos mais tarde. Esta linha deve existir em todos os arquivos QML. Observe que a versão dos módulos Qt é incluida na declaração import.

Este simples retângulo tem um identificador único, "simplebutton", que é vinculado à propriedade id. As propriedades do elemento Rectangle são relacionadas a valores, listando-se a propriedade, seguida por dois pontos (:) e então o valor. No código de exemplo, a cor cinza é relacionada à propriedade color do Rectangle. Similarmente, relacionamos as propriedades largura e a altura do Rectangle.

O elemento Text é um campo texto não editável. Nomeamos este elemento Text como buttonLabel. Para atribuir uma string como conteúdo ao campo Text, relacionamos um valor à propriedade text. O rótulo é contido dentro do Rectangle e para centraliza-lo no meio, atribuimos as âncoras do elemento Text ao seu pai, que é chamado simplebutton.

Âncoras podem ser ligadas a âncoras de outros itens, permitindo atribuições de layouts mais simples.

Devemos salvar este código como SimpleButton.qml. Executando o qmlviewer com o arquivo como argumento, será mostrado o retângulo cinza com o rótulo de texto.



Para implementar a funcionalidade de click do botão, podemos usar a manipulação de eventos do QML. Manipulação de eventos QML é muito similar ao mecanismo de signal e slot do Qt. Signals são emitidos e o slot conectado é chamado.

```
Rectangle{
    id:simplebutton
    ...

    MouseArea{
        id: buttonMouseArea

        anchors.fill: parent //ancora todos os lados da área do mouse à do retângulo
        //onClicked manipula cliques válidos no botão do mouse
        onClicked: console.log(buttonLabel.text + " clicked" )
    }
}
```

Incluimos um elemento MouseArea em nosso simplebutton. Elementos MouseArea descrevem a área interativa onde os movimentos do mouse são detectados. Para nosso botão, ancoramos toda a MouseArea ao seu pai, que é simplebutton. A sintaxe "anchors.fill" é uma forma de acessar uma propriedade especíifca chamada fill dentro de um grupo de propriedades chamado anchors. QML usa layouts baseados em âncoras onde itens podem ser ancorados a outro item, criando layouts robustos.

A MouseArea tem muitos manipuladores de sinais que são chamados durante os movimentos do mouse dentro dos limites especificados da MouseArea. Um deles é "onClicked" e é chamado sempre que um botão aceitável do mouse é clicado, o clique esquerdo sendo o default. Nós podemos vincular ações para o manipulador onClicked. Em nosso exemplo, "console.log()" mostra texto sempre que se clica na área do mouse. A função "console.log()" é uma ferramenta útil para fins de depuração e para mostrar texto.

O código no "SimpleButton.qml" é suficiente para mostrar um botão na tela e exibir o texto "clicked" quando o botão for clicado com o mouse.

```
Rectangle {
      id:Button
      ...

      property color buttonColor: "lightblue"
      property color onHoverColor: "gold"
      property color borderColor: "white"

      signal buttonClick()
      onButtonClick: {
          console.log(buttonLabel.text + " clicked" )
      }

      MouseArea{
          onClicked: buttonClick()
          hoverEnabled: true
          onEntered: parent.border.color = onHoverColor
          onExited:  parent.border.color = borderColor
      }
      // determina a cor do botão pelo uso o operador condicional
      color: buttonMouseArea.pressed ? Qt.darker(buttonColor, 1.5) : buttonColor
}
```

Um botão completamente funcional está no arquivo Button.qml. Os trechos de código neste artigo têm algum código omitido, denotados por reticências ou porque eles foram introduzidos em seções anteriores ou irrelevantes para a discussão do código atual.

Propriedades personalizadas são declaradas usando a sintaxe de nome do tipo da propriedade. No código, a propriedade buttonColor, do tipo color, é declarada e vinculada ao valor "lightblue".

A propriedade buttonColor é mais tarde usada em uma operação condicional para determinar a cor de preenchimento do botão. Note que a atribuição de valor da propriedade é possível usando sinais de igualdade, além do vínculo de valor usando o caractere dois pontos (:). Propriedades personalizadas permitem que itens internos sejam acessíveis fora do escopo do Rectangle. Existem tipos QML básicos como int, string, real, também como um tipo chamado variant.

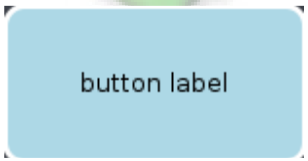
Vinculando os manipuladores de sinais "onEntered" e "onExited" a cores, a borda do botão se tornará amarela quando a seta do mouse passar por sobre o botão e reverterá a cor quando a seta do mouse deixar esta área.

Um sinal "buttonClick" é declarado em "Button.qml" colocando a palavra-chave signal na frete no nome do sinal. Todos os sinais têm manipuladores automaticamente criados, com seus nomes começando com "on". Como resultado, o "onButtonClick" é o manipulador de "buttonClick". O "onButtonClick" é então associado a uma ação a ser executada.
Em nosso botão exemplo, o manipulador do mouse "onClicked" irá simplesmente chamar "onButtonClick", que mostra um texto. O "onButtonClick" habilita objetos externos a acessarem a área do mouse do "Button" facilmente.

Por exemplo, itens podem ter mais do que uma declaração de "MouseArea" e um sinal "buttonClick" pode fazer melhor a distinção entre os vários manipuladores de sinal "MouseArea".

Agora temos o conhecimento básico para implementar itens no QML que podem manipular movimentos básicos do mouse. Criamos um rótulo "Text" dentro de um "Rectangle", com propriedades customizadas, e comportamentos implementados que respondem aos movimentos do mouse. Esta idéia de criar elementos dentro de elementos repete-se ao longo da aplicação do editor de texto.

Este botão não é prático até que seja usado como um componente para executar uma ação. Na próxima seção, criaremos um menu contendo muitos destes botões.



Criando uma página de Menu

Até esta etapa, cobrimos como criar elementos e atribuir comportamentos dentro de um único arquivo QML. Nesta seção, cobriremos como importar elementos QML e como reutilizar alguns dos componentes criados para construir outros componentes.

Menus exibem o conteúdo de uma lista, cada item tendo a capacidade de executar uma ação. Em QML, podemos criar um menu de formas diferentes. Primeiro, criamos um menu contendo botões que eventualmente executar diferentes ações. O código do menu está em "FileMenu.qml".

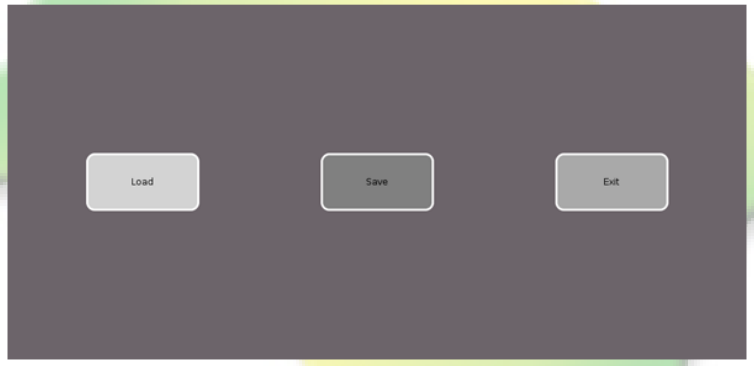
```
import QtQuick 1.0                  \\importa o módulo principal Qt QML
import "folderName"                 \\importa o conteúdo da pasta
import "script.js" as Script        \\importa um arquivo Javascript e o nomeia como Script
```

A sintaxe mostrada acima mostra como usar a palavra-chave import. Isto é requerido para usar arquivos Javascript, ou arquivos QML que não estão dentro do mesmo diretório.
Desde que "Button.qml" esteja no mesmo diretório que "FileMenu.qml", não precisamos importar o arquivo "Button.qml" para utilizá-lo. Podemos criar um elemento "Button" declarando "Button{}", similiar à declaração "Rectangle{}".

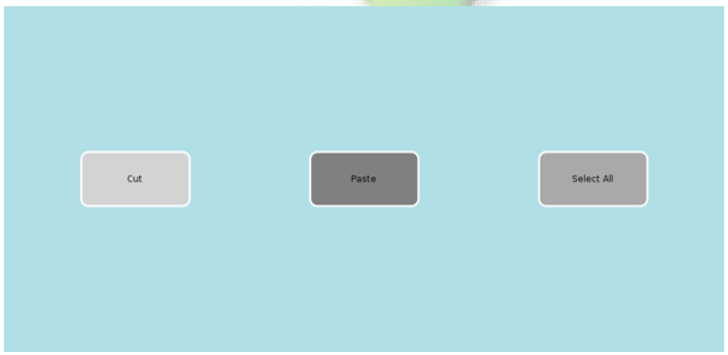
No arquivo "FileMenu.qml":

```
Row{
    anchors.centerIn: parent
    spacing: parent.width/6

    Button{
        id: loadButton
        buttonColor: "lightgrey"
        label: "Load"
    }
    Button{
        buttonColor: "grey"
        id: saveButton
        label: "Save"
    }
    Button{
        id: exitButton
        label: "Exit"
        buttonColor: "darkgrey"

        onButtonClick: Qt.quit()
    }
}
```

Em "FileMenu.qml", nós declaramos três elementos "Button". Eles são declarados dentro de um elemento "Row", um posicionador que irá posicionar seus filhos ao longo de uma linha vertical.
A declaração "Button" reside em "Button.qml", que é o mesmo "Button.qml" que nós usamos na seção anterior. Novos vínculos de propriedades podem ser declaradas dentro de novos botões criados, sobrescrevendo efetivamente as propriedades "setadas" em "Button.qml". O botão chamado "exitButton" irá sair e fechar a janela quando clicado. Observe que o manipulador de sinal "onButtonClick" em "Button.qml" será chamado além do manipulador "onButtonClick" em "exitButton".



A declaração "Row" é feita em um "Rectangle", criando um contêiner retângulo para uma linha de botões. Este retângulo adicional cria uma forma indireta de organizar a linha de botões dentro do um menu.
A declaração do menu "edit" é muito similar nesta fase. O menu tem botões que possuem rótulos: Copiar, Colar e Selecionar tudo.



Armado com nossos conhecimentos de importação e personalização de componentes previamnte feitos, podemos combinar estas páginas de menu para criar uma barra de menu, consistindo de botões para selecionar o menu, e ver como podemos estruturar dados usando QML.

Implementando uma Barra de Menu

Nossa aplicação de editor de textos precisará ter uma forma de mostrar menus usando uma barra de menu. A barra de menu irá selecionar os diferentes menus e o usuário poderá escolher qual menu exibir.
Seleção de menu implica em que os menus precisem de mais estrutura do que meramente exibi-los em uma linha. QML usa models (modelos) e views (visualizações) para estruturar dados e mostrar os dados estruturados.

Usando Data Models e Views

QML tem diferentes visualizadores de dados (data views) que mostram modelos de dados (data models). Nossa barra de menu mostará os menus em uma lista, com um cabeçalho que mostra uma linha de nomes de menus. A lista de menus é declarada dentro de um VisualItemModel. O elemento VisualItemModel contem itens que já tem visualizações como um elemento Rectangle e elementos de interface gráfica importados. Outros tipos de modelos como o elemento ListModel precisam de uma view para mostrar seus dados.

Declaramos dois itens visuais no "menuListModel", o "FileMenu" e o "EditMenu". Personalizamos os dois menus e os exibimos usando uma "ListView". O arquivo "MenuBar.qml" contem a declaração QML e um menus simples edit é definidoo em "EditMenu.qml".

```
VisualItemModel{
    id: menuListModel
    FileMenu{
        width: menuListView.width
        height: menuBar.height
        color: fileColor
    }
    EditMenu{
        color: editColor
        width:  menuListView.width
        height: menuBar.height
    }
}
```

O elemento ListView mostrará um modelo de acordo com um representante (data view).

```
ListView{
    id: menuListView

    //Âncoras são definidas para reagir Anchors às âncoras da janela
    anchors.fill:parent
    anchors.bottom: parent.bottom
    width:parent.width
    height: parent.height

    //model contém os dados
    model: menuListModel

//controla o movimento de mudança do menu
```

```
        snapMode: ListView.SnapOneItem
        orientation: ListView.Horizontal
        boundsBehavior: Flickable.StopAtBounds
        flickDeceleration: 5000
        highlightFollowsCurrentItem: true
        highlightMoveDuration:240
        highlightRangeMode: ListView.StrictlyEnforceRange
    }
```

Adicionalmente, "ListView" herda de "Flickable", fazendo a lista responder a objetos arrastados pelo mouse e outros movimentos. A última parte do código acima "seta" as propriedades para criar o movimento de flick desejado à nossa visualização. Em particular, a propriedade "highlightMoveDuration" altera a duração da transição flick. Um valor de "highlightMoveDuration" mais alto resulta em uma mudança de menu mais lenta.

O "ListView" mantém os itens do modelo através de um índice e cada item visual no modelo é acessível através do index, na ordem da declaração. Mudar o "currentIndex" efetivamente muda o item destacado na "ListView". O cabeçalho de nossa barra de menu exemplifica este efeito. Existem dois botões em uma linha, ambos mudando o menu corrente quando clicados. O "fileButton" muda o menu corrente para o menu file quando clicado, o índice sendo 0 (zero) porque FileMenu é declarado primeiro em "menuListModel". Similarmente, o "editButton" mudará o menu corrente para o "EditMenu" quando clicado.

O retângulo "labelList" tem um valor "z" de "1", denotando que ele é mostrado na frente da barra de menu. Items com maior valor de z são mostrados na frente daqueles itens com valores z mais baixos. O valor default de z é 0 (zero).

```
Rectangle{
    id: labelList
    ...
    z: 1
    Row{
        anchors.centerIn: parent
        spacing:40
        Button{
            label: "File"
            id: fileButton
            ...
            onButtonClick: menuListView.currentIndex = 0
        }
        Button{
            id: editButton
            label: "Edit"
            ...
            onButtonClick:    menuListView.currentIndex = 1
        }
    }
}
```

A barra de menu que acabamos de criar pode ser movimentada para acessar os menus pelo clique nos nomes dos menus no topo. A troca de telas de menu parece intuitiva e responsiva.

Montando o editor de textos

Declarando uma TextArea

Nosso editor de textos não é um editor de textos se não contiver uma área de texto editável. Elementos "TextEdit" QML permitem a declaração de um área de texto editável multilinha.
"TextEdit" é diferente de um elemento Text, que não permite ao usuário editar diretamente o texto.

```
TextEdit{
    id: textEditor
    anchors.fill:parent
    width:parent.width; height:parent.height
    color:"midnightblue"
    focus: true

    wrapMode: TextEdit.Wrap

    onCursorRectangleChanged: flickArea.ensureVisible(cursorRectangle)
}
```

O editor tem sua propriedade font color definida e está setado para "quebrar" o texto. O "TextEdit" está dentro de uma área flickable que rolará o texto se o cursor do texto estiver fora da área visível. A função "ensureVisible()" irá verificar se o retângulo do cursor está fora dos limites visíveis e moverá a área de texto de acordo. QML usa sintaxe Javascript para seus scripts, e como previamente mencionado, arquivos Javascript podem ser importados e usados dentro de um arquivo QML.

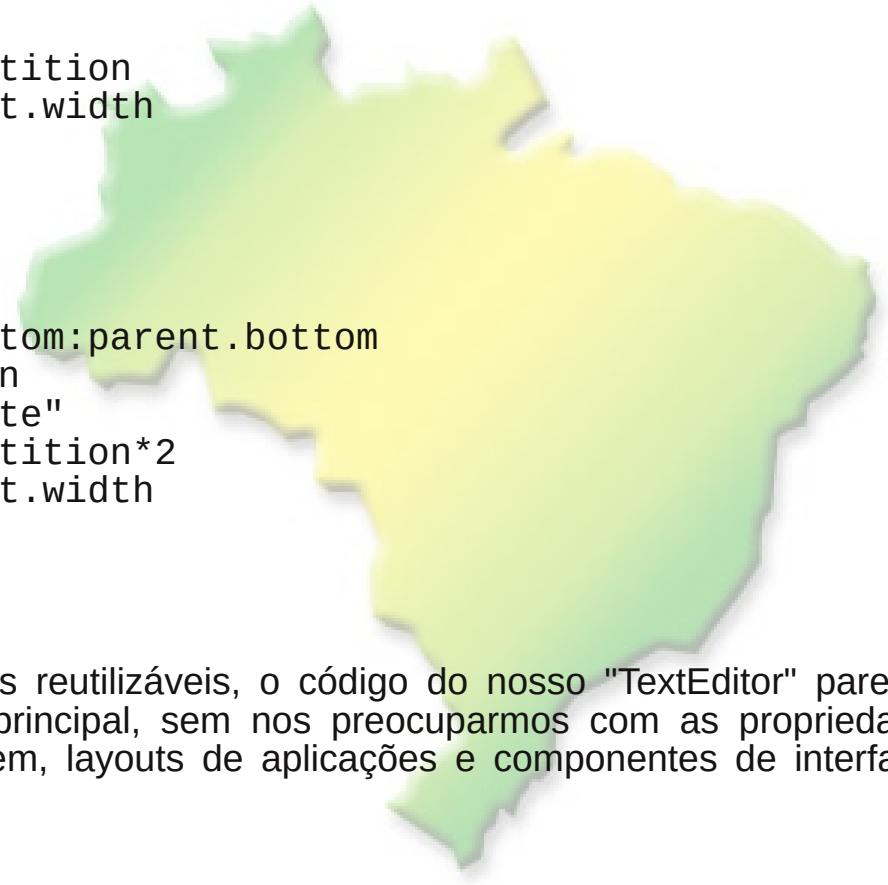
```
function ensureVisible(r){
        if (contentX >= r.x)
            contentX = r.x;
        else if (contentX+width <= r.x+r.width)
            contentX = r.x+r.width-width;
        if (contentY >= r.y)
            contentY = r.y;
        else if (contentY+height <= r.y+r.height)
            contentY = r.y+r.height-height;
 }
```

Combinando componentes para o Editor de Textos

Agora estamos prontos para criar o layout do nosso editor de textos usando QML. O editor de textos tem dois componentes, a barra de menus que nós criamos e a área de texto. QML permite-nos reutilizar componentes, simplificando o nosso código portanto, pela importação de componentes e personalização quando necessário. Nosso editor de textos divide a janela em duas; um terço da tela é dedicado à barra de menu e dois terços da tela mostram a área de texto. A barra de menu é mostrada na frente de quaisquer outros elementos.

```
    Rectangle{
        id: screen
        width: 1000; height: 1000

        //a tela é particionada entre MenuBar e TextArea. 1/3 da tela é atribuido ao
MenuBar
```

```
        property int partition: height/3

        MenuBar{
            id:menuBar
            height: partition
            width:parent.width
            z: 1
        }

        TextArea{
            id:textArea
            anchors.bottom:parent.bottom
            y: partition
            color: "white"
            height: partition*2
            width:parent.width
        }
    }
```



Pela importação de componentes reutilizáveis, o código do nosso "TextEditor" parece muito mais simples. Podemos então personalizar a aplicação principal, sem nos preocuparmos com as propriedades que já têm comportamentos definidos. Usando esta abordagem, layouts de aplicações e componentes de interface gráfica de usuário podem ser criados facilmente.

Decorando o Editor de Textos

Implementando uma Interface de Desenhador (ui...)

Nosso editor de textos parece simples e nós precisamos decorá-lo. Usando QML, podemos declarar transições e animar nosso editor de textos. Nossa barra de menu está ocupado um terço da tela e seria bom se ela aparecesse apenas quando nós quiséssemos.

Podemos adicionar uma interface de desenhador, que irá contrair ou expandir a barra de menu quando clicada. Em nosso implementação, temos um retângulo fino que responde aos cliques do mouse.
O desenhador, assim como a aplicação, tem dois estados (states): o estado "desenhador aberto" e o estado "desenhador fechado". O item desenhador é uma tira retangular com uma pequena altura. Existe um elemento Image aninhado declarando que um ícone de seta será centralizado dentro do desenhador. O desenhador atribui um estado a toda a aplicação, com o identificador "screen", sempre que o usuário clique na área do mouse.


```
import Qt 4.7

Rectangle{
        id:textArea

        function paste(){ textEdit.paste()}
        function copy() { textEdit.copy() }
        function selectAll() { textEdit.selectAll() }

        width :400; height:400

        property color fontColor: "white"
        property alias textContent: textEdit.text
        Flickable{
                id: flickArea
                width: parent.width; height: parent.height
                anchors.fill:parent

                boundsBehavior: Flickable.StopAtBounds
                flickableDirection: Flickable.HorizontalFlick
                interactive: true
                //Will move the text Edit area to make the area visible when scrolled with keyboard strokes
                function ensureVisible(r){
                        if (contentX >= r.x)
                        contentX = r.x;
                        else if (contentX+width <= r.x+r.width)
                        contentX = r.x+r.width-width;
                        if (contentY >= r.y)
                        contentY = r.y;
                        else if (contentY+height <= r.y+r.height)
                        contentY = r.y+r.height-height;
                }
        Flickable{
                id: flickArea
                width: parent.width; height: parent.height
                anchors.fill:parent

                boundsBehavior: Flickable.StopAtBounds
                flickableDirection: Flickable.HorizontalFlick
                interactive: true
                //Will move the text Edit area to make the area visible when scrolled with keyboard strokes
                function ensureVisible(r){
                        if (contentX >= r.x)
```

```
Rectangle{
    id:drawer
    height:15

    Image{
        id: arrowIcon
        source: "images/arrow.png"
        anchors.horizontalCenter: parent.horizontalCenter
    }

    MouseArea{
        id: drawerMouseArea
        anchors.fill:parent
        onClicked:{
            if (screen.state == "DRAWER_CLOSED"){
                screen.state = "DRAWER_OPEN"
            }
            else if (screen.state == "DRAWER_OPEN"){
                screen.state = "DRAWER_CLOSED"
            }
        }
        ...
    }
}
```

Um estado é simplesmente uma coleção de configurações e é declarado em um elemento "State". Uma lista de estados pode ser listada e vinculada às propriedades states. Em nossa aplicação, os dois estados são chamados DRAWER_CLOSED e DRAWER_OPEN. Configurações de item são declaradas em elementos PropertyChanges. No estado DRAWER_OPEN, existem quatro itens que receberão mudanças de propriedades. O primeiro alvo, menuBar, mudará suas propriedades y para 0 (zero). Similarmente, o "textArea" será abaixada pra uma nova posição quando o

estado for DRAWER_OPEN. A "textArea", o "desenhadador", e o ícone do desenhador sofrerão alterações de propriedades para atender ao estado atual.

```
      states:[
          State {
              name: "DRAWER_OPEN"
              PropertyChanges { target: menuBar; y: 0}
              PropertyChanges { target: textArea; y: partition + drawer.height}
              PropertyChanges { target: drawer; y: partition}
              PropertyChanges { target: arrowIcon; rotation: 180}
          },
          State {
              name: "DRAWER_CLOSED"
              PropertyChanges { target: menuBar; y:-height; }
              PropertyChanges { target: textArea; y: drawer.height; height: screen.height
- drawer.height }
              PropertyChanges { target: drawer; y: 0 }
              PropertyChanges { target: arrowIcon; rotation: 0 }
          }
      ]
```

Mudanças de estado são abruptas e precisam de transições mais suaves. Transições entre estados são definidas usando o elemento Transition, que pode ser vinculado às propriedades "transitions" do item.
Nosso editor de textos tem uma transição de estado sempre que o estado muda para DRAWER_OPEN ou DRAWER_CLOSED. Importante: a transição precisa de um estado from (de) e um estado to (para)
mas para nossas transições, podemos usar o símbolo coringa * para denotar que a transição se aplica a todas as mudanças de estado.

Durante transições, podemos atribuir animações às mudanças de propriedades. Nosso menuBar troca de posição de y:0 para y:-partition e podemos animar esta transição usando o elemento "NumberAnimation". Declaramos que a

propriedade target será animada por uma certa duração de tempo e usando uma certa curva de atenuação. Uma curva de atenuação controle a taxa de animação e comportamento de interpolação durante transições de estado. A curva de atenuação que escolhemos é Easing.OutQuint, que torna mais lento o movimento próximo ao fim da animação.

```
      transitions: [
          Transition {
             to: "*"
             NumberAnimation { target: textArea; properties: "y, height"; duration: 100;
easing.type:Easing.OutExpo }
             NumberAnimation { target: menuBar; properties: "y"; duration: 100;
easing.type: Easing.OutExpo }
             NumberAnimation { target: drawer; properties: "y"; duration: 100;
easing.type: Easing.OutExpo }
          }
      ]
```

Outra forma de animar mudanças de propriedades é declarar um elemento "Behavior". Uma transição apenas funciona durante mudanças de estado e "Behavior" pode definir uma animação para uma mudança geral de propriedade. No editor de textos, a seta tem um "NumberAnimation" animando sua propriedade "rotation" sempre que a propriedade mudar.

Em TextEditor.qml:

```
      Behavior{
          NumberAnimation{property: "rotation";easing.type: Easing.OutExpo }
      }
```

Voltando aos nossos componentes com conhecimento de estados e animações, podemos melhorar a aparência dos componentes. Em Button.qml, podemos adicionar mudanças das proprieadades "color" e "scale" quando o botão for clicado. Tipos "Color" são animados usando "ColorAnimation" e numbers são animados usando "NumberAnimation". A sintaxe "on nomeDaPropriedade" mostrada abaixo é útil quando fazemos referência a uma simples propriedade.

Em Button.qml:

```
...

color: buttonMouseArea.pressed ? Qt.darker(buttonColor, 1.5) : buttonColor
Behavior on color { ColorAnimation{ duration: 55} }

scale: buttonMouseArea.pressed ? 1.1 : 1.00
Behavior on scale { NumberAnimation{ duration: 55} }
```

Além disso, podemos melhorar a aparência de nossos componentes QML adicionando efeitos de cores como gradientes e efeitos de opacidade. Declarando um elemento "Gradiente" sobreporá a propriedade "color" do elemento. Você pode declarar uma cor no gradiente usando o elemento "GradientStop". O gradiente é posicionado usando uma escala, entre 0.0 e 1.0.

Em MenuBar.qml

```
gradient: Gradient {
    GradientStop { position: 0.0; color: "#8C8F8C" }
    GradientStop { position: 0.17; color: "#6A6D6A" }
    GradientStop { position: 0.98;color: "#3F3F3F" }
    GradientStop { position: 1.0; color: "#0e1B20" }
}
```

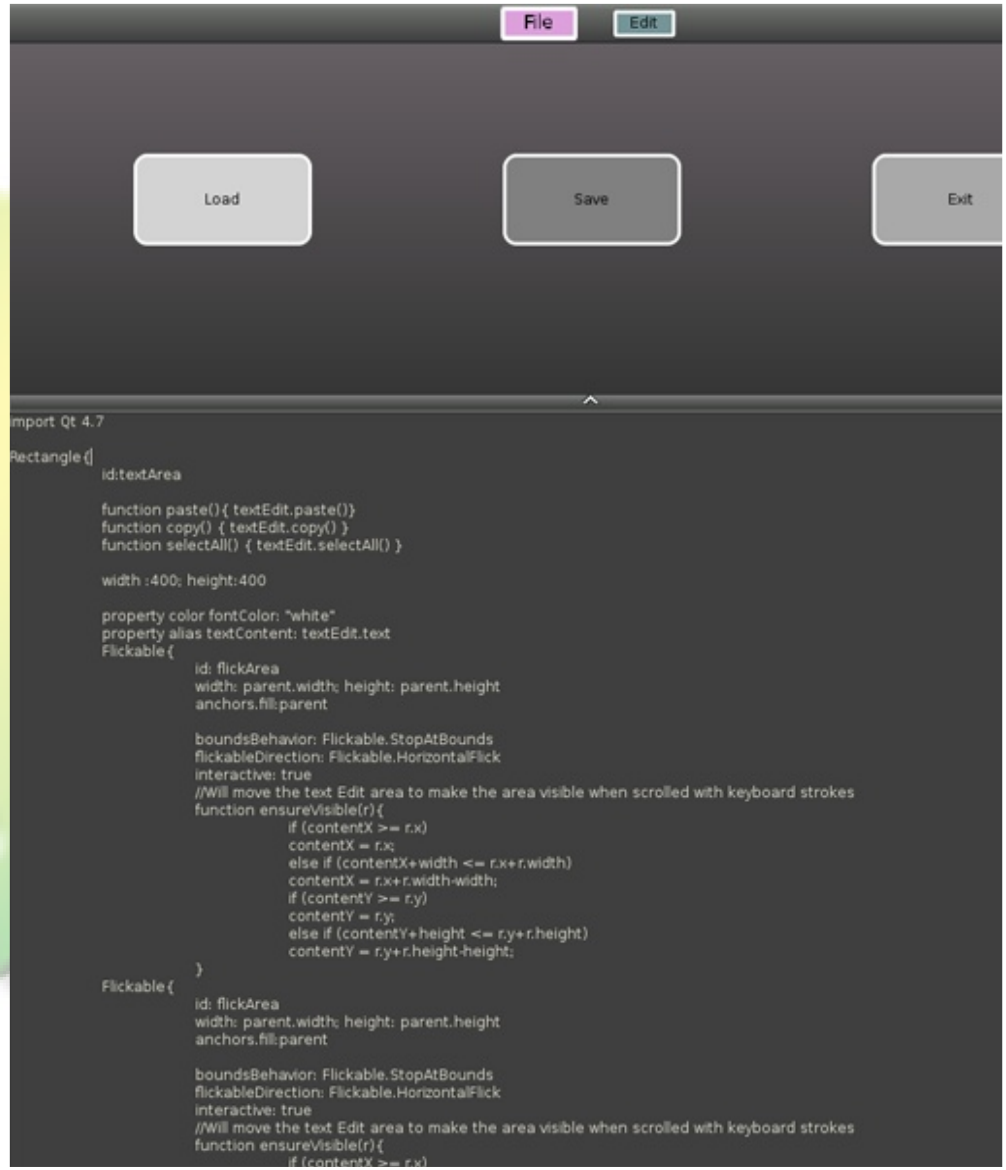
O gradiente é usado pela barra de menu para mostrar um gradiente simulando profundidade. A primeira cor começa em 0.0 e a última cor está em 1.0.

Aonde ir partindo daqui.

Terminamos de construir a interface de usuáro de uma editor de textos muito simples. Daqui pra frente, a interface de usuário está completa e podemos implementar a lógica da aplicação usando Qt e C++ comum. QML trabalha perfeitamente como uma ferramenta de prototipagem, separando a lógica da aplicação do desenho da Interface de Usuário.

Extendendo QML usando Qt e C++

Agora que temos o layout de nosso editor de textos, podemos implementar as funcionalidades do editor em C++. Usando QML com C++ permite-nos criar nossa lógica da aplicação usando Qt. Podemos criar um contexto QML na aplicação C++ usando as classes Declarative de Qt e mostrar o elemento QML usando um Graphics Scene. Alternativamente, podemos exportar nosso código C++ em um plugin que pode ser lido pela ferramenta qmlviewer. Para nossa aplicação, implementaremos as funções "load" e "save" em C++ e as exportaremos como um plugin.
Desta forma, apenas precisamos carregar o arquivo QML diretamente ao invés de rodar um executável.



Expondo classes C++ para QML

Implementaremos a carga e o salvamento de arquivos usando Qt e C++. Classes C++ e funções podem ser usadas em QML registrando-as. A classe também precisa ser compilada como um plugin Qt e o arquivo QML precisa saber onde o plugin está localizado.

Para nossa aplicação, precisamos criar os seguintes itens:

1. Clase Directory que manipulará operações relacionadas a diretórios
2. Classe File que é um QObject, simulando a lista de arquivos em um diretório
3. Classe plugin que registrará a classe para o contexto QML
4. Arquivo de projeto Qt que irá compilar o plugin
5. Um arquivo qmldir informando à ferramenta qmlviewer onde encontrar o plugin

Montando um plugin Qt

Para montar um plugin, precisamos definir o seguinte em um projeto Qt. Primeiro, os fontes necessários, cabeçalhos (headers), e módulos Qt precisam ser adicionados ao nosso arquivo de projeto. Todo o código C++ e arquivos de projeto estão no diretório "filedialog".

Em cppPlugins.pro:

```
TEMPLATE = lib
CONFIG += qt plugin
QT += declarative

DESTDIR +=  ../plugins
OBJECTS_DIR = tmp
MOC_DIR = tmp
```

```
TARGET = FileDialog

HEADERS +=     directory.h \
        file.h \
        dialogPlugin.h

SOURCES +=    directory.cpp \
        file.cpp \
        dialogPlugin.cpp
```

Em particular, compilamos Qt com o módulo declarativo e o configuramos como um plugin, necessitando de um template "lib". Colocaremos o plugin compilado dentro do diretório de plugins pai.

Registrando uma classe no QML

Em dialogPlugin.h:

```
#include <QtDeclarative/QDeclarativeExtensionPlugin>

class DialogPlugin : public QDeclarativeExtensionPlugin
{
    Q_OBJECT

    public:
    void registerTypes(const char *uri);

};
```

Nossa classe plugin, DialogPlugin é uma subclasse de QDeclarativeExtensionPlugin. Precisamos implementar a função

herdada registerTypes(). O arquivo dialogPlugin.cpp parece com isto:

DialogPlugin.cpp:

```
#include "dialogPlugin.h"
#include "directory.h"
#include "file.h"
#include <QtDeclarative/qdeclarative.h>

void DialogPlugin::registerTypes(const char *uri){

    qmlRegisterType<Directory>(uri, 1, 0, "Directory");
    qmlRegisterType<File>(uri, 1, 0,"File");
}

Q_EXPORT_PLUGIN2(FileDialog, DialogPlugin);
```

A função "registerTypes()" registra nossas classes "File" e "Directory" no QML. Esta função precisa do nome da classe para seu template, um número maior de versão, um número menor de versão, e um nome para nossa classe.

Precisamos exportar o plugin usando a macro Q_EXPORT_PLUGIN2. Observe que em nosso arquivo dialogPlugin.h, temos a macro Q_OBJECT no topo de nossa classe. Também precisamos executar qmake no arquivo de projeto para gerar o código meta-objeto necessário .

Criando propriedades em uma classe C++

Podemos criar elementos QML e propriedades usando C++ e o sistema de meta-objetos de Qt. Podemos implementar propriedades usando slots e signals, fazendo Qt tomar conhecimento destas propriedades.

Estas propriedades podem ser usada em QML.

Para o editor de textos, precisamos ser capazes de carregar e salvar arquivos. Tipicamente, estes recursos estão contidos em um diálogo de arquivo (file dialog). Felizmente, podemos usar "QDir", "QFile" e "QTextStream" para implementar leitura de diretórios e fluxos (streams) de entrada e saída.

```
      class Directory : public QObject{

          Q_OBJECT

          Q_PROPERTY(int filesCount READ filesCount CONSTANT)
          Q_PROPERTY(QString filename READ filename WRITE setFilename NOTIFY
filenameChanged)
          Q_PROPERTY(QString fileContent READ fileContent WRITE setFileContent NOTIFY
fileContentChanged)
          Q_PROPERTY(QDeclarativeListProperty<File> files READ files CONSTANT )

          ...
```

A classe "Directory" usa o Sistema de "Meta-Objeto" de Qt para registrar propriedades necessárias para realizar manipulação de arquivos. A classe "Directory" é exportada como um plugin e é utilizável em QML como um elemento "Directory". Cada uma das propriedades listadas usando a macro Q_PROPERTY é uma propriedade QML.

A macro Q_PROPERTY declara uma propriedade e também suas funções de leitura (read) e escrita (write) no Sistema de Meta-Objetos do Qt. Por exemplo, a propriedade filename, do tipo "QString" pode ser lida usando a função filename() e escrita usando a função "setFileName()". Adicionalmente, existe um signal associado à propriedade filename chamado "filenameChanged()", que é emitido sempre que a propriedade muda. As funções de leitura e escrita são declaradas como públicas no arquivo de cabeçalho.

De modo semelhante, temos as outras propriedades declaradas de acordo com seus usos. A propriedade "filesCount" indica o número de arquivos em um diretório. A propriedade "filename" é definida para o nome do arquivo selecionado e

o arquivo carregado/salvo é armazenado na propriedade "fileContent".

```
Q_PROPERTY(QDeclarativeListProperty<File> files READ files CONSTANT )
```

A propriedade files list é uma lista de todos os arquivos filtrados em um diretório. A classe Directory é implementada para filtrar arquivos inválidos de texto; apenas arquivos com uma extensão .txt são válidos. Além disse, QLists podem ser usados em arquivos QML declarando-os como uma QDeclarativeListProperty em C++. O obeto no template precisa herdar de um QObject, portanto, a classe File deve também herdar de QObject. Na clasee Directory, a lista de objetos File é armazanada em uma QList chamada m_fileList.

```
class File : public QObject{

    Q_OBJECT
    Q_PROPERTY(QString name READ name WRITE setName NOTIFY nameChanged)

    ...
};
```

As propriedades podem ser usadas em QML como parte das propriedades de elementos de Directory. Observe que não temos que criar um identificador com a propriedade id em nosso código C++.

```
Directory{
    id: directory

    filesCount
    filename
    fileContent
    files
```

```
        files[0].name
    }
```

Como arquivos QML usam sintaxe e estrutura Javascript, podemos iterar através da lista de arquivos e recuperar suas propriedades. Para recuperar o primeira propriedade "name" do arquivo, nós podemos chamar files[0].name.

Funções comuns C++ são também acessíveis a partir do QML. As funções de carregar e salvar o arquivo são implementadas em C++ e declaradas usando a macro Q_INVOKABLE. Alternativamente, podemos declarar as funções como slots e as funções estarão acessíveis a partir do QML.

Em Directory.h:

```
    Q_INVOKABLE void saveFile();
    Q_INVOKABLE void loadFile();
```

A classe Directory também tem que notificar outros objetos sempre que o conteúdo do diretório mude. Esta característica é executada como um signal. Como foi dito antes, "signals" QML tem um manipulador correspondente com seus nomes prefixados com "on". O signal é chamado directoryChanged e ele é emitido sempre que há uma atualização de um diretório. A atualização simplesmente recarrega o conteúdo do diretório e atualiza a lista de arquivos válidos no diretório. Itens QML podem então ser notificados anexando uma ação ao manipulador de sinais "onDirectoryChanged".

As propriedades list precisam ser exploradas futuramente. Isto porque propriedades list usam "callbacks" para acessar o conteúdo da lista. A propriedade list é do tipo "QDeclarativeListProperty<File>". Sempre que a lista é acessada, a função acessora precisa retornar um QDeclarativeListProperty<File>. O tipo template File, precisa ser um derivado de QObject. Além disso, para criar um QDeclarativeListProperty, o acessor da lista e modificadores precisam ser passados para o contrutor como ponteiros para funções.

A lista, um QList em nosso caso, também precisa ser uma lista de ponteiros para File.

O construtor de QDeclarativeListProperty e a implementação de Directory:

```
QDeclarativeListProperty  ( QObject * object, void * data, AppendFunction append,
CountFunction count = 0, AtFunction at = 0, ClearFunction clear = 0 )
QDeclarativeListProperty<File>( this, &m_fileList, &appendFiles, &filesSize, &fileAt,
 &clearFilesPtr );
```

O construtor passa ponteiros para funções que serão acrescentadas à lista, conta a lista, retorna o item usando um índice e limpa a lista. Apenas a função acrescentada é obrigatória. Observe que o ponteiro para a função deve combinar com a definição de AppendFunction, CountFunction, AtFunction ou ClearFunction.

```
void appendFiles(QDeclarativeListProperty<File> * property, File * file)
File* fileAt(QDeclarativeListProperty<File> * property, int index)
int filesSize(QDeclarativeListProperty<File> * property)
void clearFilesPtr(QDeclarativeListProperty<File> *property)
```

Para simplificar nosso diálogo de arquivo, a classe Directory filtra arquivos texto inválidos, que são arquivos sem a extensão .txt. Se um nome de arquivo não tiver a extensão ".txt", então ele não deve ser visto em nosso diálogo. Também, a implementação certifica que os arquivos salvos tenham a extensão .txt no nome. Directory usa "QTextStream" para ler o arquivo e enviar o conteúdo para um arquivo.

Com nosso elemento Directory, podemos recuperar os arquivos como uma lista, descobrir quantos arquivos texto existem no diretório da aplicação, pegar o nome do arquivo e conteúdo como uma string, e ser notificado sempre que ocorram mudanças no conteúdo do diretório.

Importando um Plugin no QML

A ferramenta qmlviewer importa arquivos que estão no mesmo diretório da aplicação. Podemos também criar arquivo qmldir contendo as localizações dos arquivos QML que queremos importar. O arquivo qmldir pode também armazenar localizações de plugins e outros recursos.

Em qmldir:

```
Button ./Button.qml
FileDialog ./FileDialog.qml
TextArea ./TextArea.qml
TextEditor ./TextEditor.qml
EditMenu ./EditMenu.qml

plugin FileDialog plugins
```

O plugin que acabamos de criar é chamado "FileDialog", como indicado pelo campo TARGET no arquivo de projeto. O plugin compilado está no diretório de plugins.

Integrando o File Dialog com o File Menu

Nosso FileMenu precisa mostrar o elemento FileDialog, contendo uma lista de arquivos texto em um diretório permitindo assim que o usuário selecione o arquivo clicando na lista.
Precisamos ainda atribuir os botões save, load e new a suas respectivas ações. O FileMenu contém um campo text input editável para permitir ao usuário digitar um nome de arquivo usando o teclado.
O elemento Directory é usado no arquivo FileMenu.qml e notificar o elemento FileDialog que o diretório atualizou seu conteúdo. Esta notificação é executada no manipulador de signal, onDirectoryChanged.

Em FileMenu.qml:

```
Directory{
    id:directory
    filename: textInput.text
    onDirectoryChanged: fileDialog.notifyRefresh()
}
```

Mantendo a simplicidade de nossa aplicação, o diálogo de arquivo estará sempre visível e não mostrará arquivos texto inválidos, que não tem uma extensão ".txt" em seus nomes.

Em FileDialog.qml:

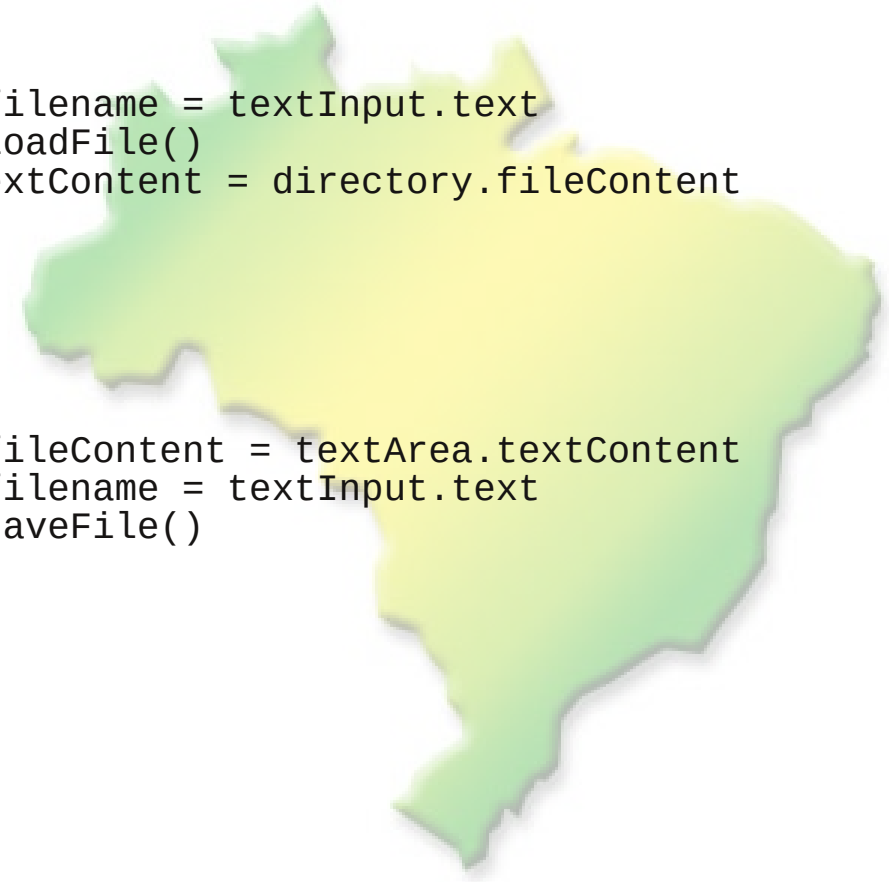
```
signal notifyRefresh()
onNotifyRefresh: dirView.model = directory.files
```

O elemento FileDialog mostrará o conteúdo de um diretório lendo sua propriedade lista, chamada files. Os arquivos são usados como o modelo de um elemento GridView, que mostra itens de dados em um grid de acordo com um "delegate" (representante). O "delegate" manipula a aparência e o modelo e nosso diálogo de arquivo simplesmente criará um grid com texto centralizado.
Clicar no nome do arquivo resultará no aparecimento de um retângulo para realçar o nome do arquivo. O FileDialog é notificado sempre que o signal "notifyRefresh" for emitido, recarregando os arquivos no diretório.

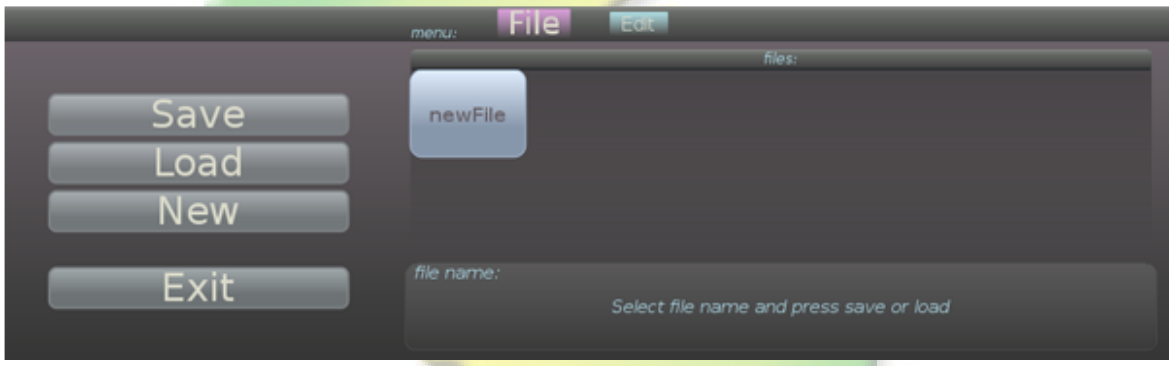
Em FileMenu.qml:

```
Button{
    id: newButton
    label: "New"
    onButtonClick:{
        textArea.textContent = ""
    }
```

```
}
Button{
    id: loadButton
    label: "Load"
    onButtonClick:{
        directory.filename = textInput.text
        directory.loadFile()
        textArea.textContent = directory.fileContent
    }
}
Button{
    id: saveButton
    label: "Save"
    onButtonClick:{
        directory.fileContent = textArea.textContent
        directory.filename = textInput.text
        directory.saveFile()
    }
}
Button{
    id: exitButton
    label: "Exit"
    onButtonClick:{
        Qt.quit()
    }
}
```



Nosso "FileMenu" pode agora conectar a suas respectivas ações. O botão saveButton irá transferir o texto do TextEdit para a propriedade fileContent do diretório, copiando então seu nome de arquivo do text input editável. Finalmente, o botão chama a função saveFile(), salvando o arquivo. O botão loadButton tem uma execução similar. Também a ação New limpará o conteúdo do "TextEdit".
Além disso, os botões de EditMenu estão conectados às funções de copiar, colar e selecionar todo o texto no editor de textos de TextEdit.

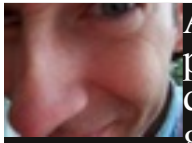


Executando o editor de textos

Precisamos compilar o plugin de diálogo de arquivos C++ antes que o editor de textos possa ser executado.
Para compilar, entre no diretório gsQml, então execute qmake e compile usando make ou nmake, dependendo da sua plataforma. Para executar, rode o qmlviewer e abra o arquivo texteditor.qml.

O código fonte está no diretório examples/tutorials/gettingStarted/gsQml da instalação do Qt.

Como pensar como um cientista da Computação
Versão C++, Primeira Edição



Allen B. Downey é professor de Ciência da Computação e gentilmente permitiu a publicação da tradução de seu livro "How to think like a computer science" na Revista Qt. O endereço do seu site é o http://allendowney.com/.

Capítulo 1
O caminho do programa

O objetivo deste livro é ensiná-lo como pensar como um cientista da computação. Eu gosto do jeito como cientistas da computação pensam porque eles combinam alguns dos melhores recursos de Matemática, Engenharia e Ciências Naturais. Como matemáticos, cientistas da computação usam linguagens formais para denotar idéias (especificamente computação). Como engenheiros, eles projetam coisas, montando componentes dentro de sistemas e avaliando compensações entre as alternativas. Como cientistas, eles observam o comportamento de sistemas complexos, formulam hipóteses e testam previsões.

A mais importante habilidade para um cientista da computação é resolver problemas. Com isso, quero dizer a habilidade de exprimir problemas, pensar criativamente sobre soluções e expressar uma solução clara e apuradamente. Como se vê, o processo de aprendizado de programação é uma excelente oportunidade para praticar as habilidades de solucionar problemas. É por isso que este capítulo de chama "O caminho do programa".

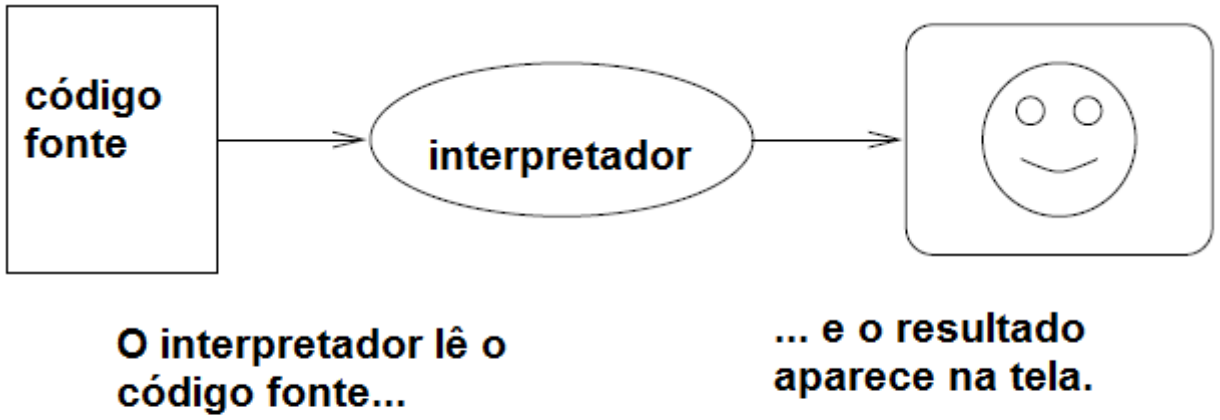
Claro que o outro objetivo deste livro é prepará-lo para o exame de Ciência da Computação da AP (http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/4483.html). Podemos não usar a abordagem mais direta para atingir este objetivo, no entanto. Por exemplo, não existem muitos exercícios neste livro que se assemelhem às questões do AP. Por outro lado, se você entender os conceitos neste livro, junto com alguns detalhes de programação em C++, você terá todas as ferramentas das quais precisa para se sair bem no exame.

1.1 - O que é uma linguagem de programação?

A linguagem de programação que você aprenderá é C++, porque esta é a linguagem na qual baseia-se o exame da AP, desde 1998. Antes disso, o exame utilizava Pascal. Ambos, C++ e Pascal são linguagens de alto nível; outras linguagems de alto nível sobre as quais você pode ouvir são Java, C e FORTRAN. Como você pode deduzir pela expressão "linguagem de alto nível", existem também linguagens de baixo nível, algumas vezes referenciadas como linguagens de máquina ou linguagens de montagem (assembly). Falando claramente, computadores podem apenas executar programas escritos em linguagens de baixo nível. Portanto, programas escritos em uma linguagem de alto nível têm que ser traduzidos antes que possam ser executados. Esta tradução toma algum tempo, o que é uma pequena desvantagem das linguagens de alto nível. Mas as vantagens são enormes. Primeiro, é muito mais fácil programar em uma linguagem de allto nível; por "fácil", quero dizer que o programa toma menos tempo para ser escrito, é menor e mais fácil de ler, e é susceptível à correção. Em segundo lugar, linguagens de alto nível são portáveis, o que significa que elas podem executar em diferentes tipos de computadores com poucas ou sem modificações. Programas de baixo nível podem ser executados apenas em um tipo de computador, e têm que ser reescritos para rodar em outro.

Devido a estas vantagens, quase todos os programas são escritos em linguagens de alto nível. Linguagens de baixo nível são usadas apenas para algumas aplicações especiais.

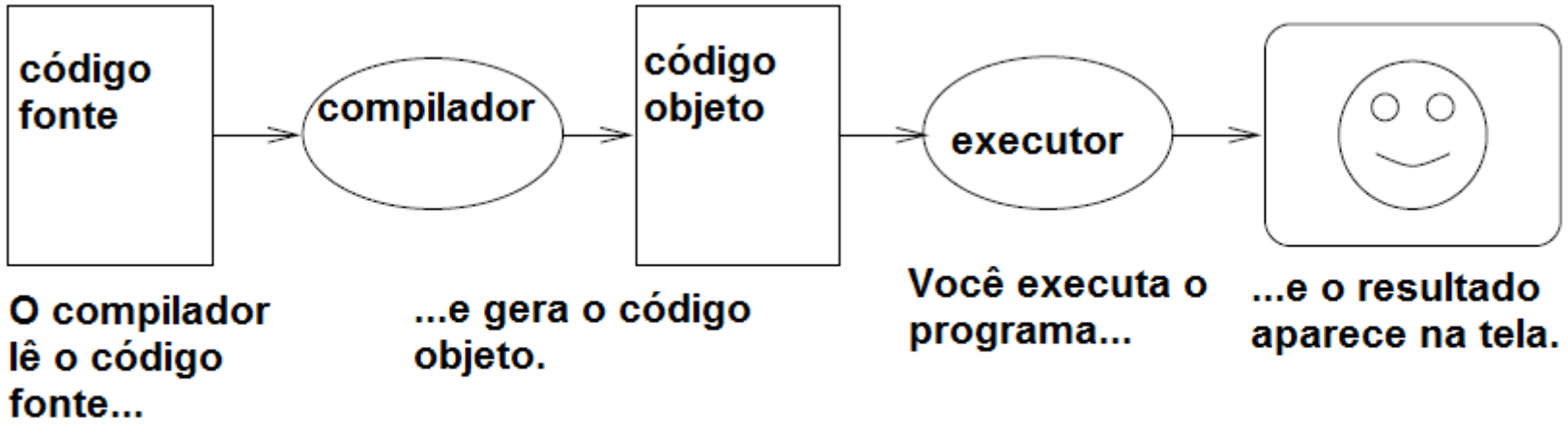
Existem duas formas de traduzir um programa: interpretando ou compilando. Um interpretador é um programa que lê um programa de alto nível, e faz o que ele diz. Com efeito, ele traduz o programa linha por linha, alternativamente lendo linhas e executando os comandos.



Um compilador é um programa que lê um programa em alto nível e traduz tudo de uma vez, antes de executar qualquer comando. Frequentemente você compila o programa em um passo separado, e então executa o código compilado depois. Neste caso, o programa de alto nível é chamado de código fonte, e o programa traduzido é chamado de código objeto ou o executável.

Como um exemplo, suponha que você escreva um programa em C++. Você usa um editor de textos para escrever o programa (um editor de textos é um simples processador de palavras). Quando o programa está pronto, você deve salva-lo em um arquivo chamado programa.cpp, onde "programa" é um nome arbitrário inventado por você, e o sufixo .cpp é uma convenção que indica
que o arquivo contém código fonte C++.

Então, dependendo de como seja o seu ambiente de desenvolvimento, você sai do editor de textos e executa o compilador. O compilador deve ler o seu código fonte, traduzi-lo, e criar um novo arquivo chamado programa.o para conter o código objeto, ou programa.exe para conter o executável.



O próximo passo é executar o programa, o que requer algum tipo de executor. O papel do executor é carregar o programa (copia-lo do disco para a memória) e fazer o computador iniciar a execução do programa. Embora este processo possa parecer complicado, a boa notícia é que na maioria dos ambientes de programação (algumas vezes chamados ambientes de desenvolvimento), estes passos são automatizados para você. Comumente você terá apenas que escrever um programa e digitar um simples comando para compilá-lo e executá-lo. Por outro lado, é útil saber quais são os passos que estão acontecendo "por trás", de modo que se algo der errado, você possa descobrir do que se trata.

O que é um programa?

Um programa é uma sequência de instruções que especificam como executar uma computação. A computação pode ser alguma coisa matemática, como resolver um sistema de equações ou encontrar as raízes de um polinômio, mas também pode ser uma computação simbólica, como procurar e substituir texto em um documento ou (estranhamente) compilar um programa. As instruções (ou comandos, ou declarações) parecem diferentes em diferentes linguagens de programação, mas existem algumas funções básicas que aparecem em quase toda linguagem de programação:

entrada: Pega dados do teclado, ou de um arquivo, ou algum outro dispositivo.

saída: Mostra dados na tela ou envia dados para um arquivo ou outro dispositivo.

cálculos matemáticos: Executa operações matemáticas básicas como adição e multiplicação.

teste: Testa determinadas condições e executa a sequência apropriada de declaraçoes. repetição: Executa alguma ação repetidamente, normalmente com alguma variação.

Acredite ou não, isto é praticamente tudo que existe a fazer. Todo programa que você já usou, não importa quão complicado, é feito de funções que se parecem mais ou menos com isso. Assim, uma forma de descrever progamação é o processo de quebrar uma grande, complexa tarefa em subtarefas cada vez menores até que finalmente a subtarefa seja simples o suficiente para executar uma destas simples funções.

1.3 O que é "debugar"?

Programar é um processo complexo, e como é realizado por seres humanos, ele frequentemente leva a erros. Por razões bizarras, erros de programação são chamados "bugs" e o processo de rastreá-los e corrigi-los é chamado "debugar".
Existem alguns tipos diferentes de erros que podem ocorrer em um programa, e é útil distinguir entre eles para rastreá-los mais rapidamente.

1.3.1 Erros em tempo de compilação

O compilador pode apenas traduzir um programa se o programa estiver sintaticamente correto; caso contrário, a compilação falha e você não estará apto a executar seu programa. A sintaxe refere-se à estrutura do seu programa e as regras sobre esta estrutura.
Por exemplo, em Inglês, uma sentença deve começar com uma letra maiúscula e terminar com um ponto. "esta sentença contém um erro de sintaxe." "Assim como esta " Para a maioria dos leitores, uns poucos erros de sintaxe não são um problema significativo, e é por isso que conseguimos ler a poesia de E. E. Cummings sem vomitar mensagens de erro. Compiladores não são tão complacentes. Se existe um único erro de sintaxe em qualquer lugar do seu programa, o compilador imprimirá uma mensagem de erro e abortará, e você não poderá executar seu programa.
Para piorar a situação, existe mais regras de sintaxe em C++ do que existem no Inglês, e as mensagens de erro que você recebe o compilador frequentemente não ajudam muito. Durante as primeiras semanas de sua carreira como programador, você provavelmente irá gastar muito tempo rastreando erros de sintaxe. Conforme você ganha experiência, no entanto, cometerá menos erros e os achará mais rapidamente.

1.3.2 Erros em tempo de execução

O segundo tipo de erro é o erro em tempo de execução, assim chamado porque o erro não aparece até a execução do programa.
Para os tipos simples de programas que escreveremos nas próximas semanas, erros em tempo de execução são raros, então levará algum tempo antes que você encontre um.

1.3.3 Erros de lógica e semântica

O terceiro tipo de erro é o de lógica e semântica. Se existe um erro de lógica em seu programa, ele compilará e executará com sucesso, no sentido de que o computador não irá gerar mensagens de erro, mas ele não fará a coisa certa. Ele fará alguma outra coisa. Especificamente, ele fará aquilo que você o mandou fazer. O problema é que o programa que você escreveu não é o programa que você queria escrever. O significado do programa (sua semântica) está errada. Identificar erros de lógica pode ser complicado, uma vez que requer que você trabalhe inversamente observando a saída do programa e tentando descobrir o que está fazendo.

1.3.4 Debug experimental

Uma das mais importantes habilidades que você precisa desenvolver no trabalho com este livro é a de debugar. Embora possa ser frustrante, debugar é um das mais ricas intelectualmente, desafiantes e interessantes partes da tarefa de programar. De certa forma, debugar é como trabalho de detetive. Você está diante de pistas e tem que deduzir os processos e eventos que levaram ao resultado
que está vendo.

Debugar também é como uma ciência experimental. Uma vez que você tenha uma idéia do que está errado, você pode modificar o seu programa e tentar novamente. Se sua hipótese estava correta, então você pode prever o resultado da alteração, e você está um passo mais perto de um programa funcionando. Se sua hipótese estava errada, você tem tem que aparecer com uma nova. Como Sherlock Holmes pregava, "Quando você tiver eliminado o impossível, o que quer que sobre, por mais improvável, deve ser a verdade." (de O Signo dos Quatro de A. Conan Doyle).

Para algumas pessoas, programar e debugar são a mesma coisa. Ou seja, programar é o processo de debugar gradualmente um programa até que ele faça o que você quer. A idéia é que você deve sempre começar com um programa funcional que faça alguma coisa, e faça pequenas modificações, debugando-as enquanto avança, de modo que você sempre tenha um programa funcional.

Por exemplo, o Linux é um sistema operacional que contém milhares de linhas de código, mas começou como um simples programa usado por Linus Torvalds para explorar o chip Intel 80386. De acordo com Larry Greenfield, "Um dos primeiros projetos de Linus era um programa que alternaria entre a impressão de AAAA e BBBB. Mais tarde, isto evoluiu para o Linux" (do livro "The Linux Users' Guide Beta Version 1).
Em capítulos futuros darei mais sugestões sobre debug e outras práticas de programação.

1.4 Linguagens formais e naturais

Linguagens naturais são aquelas faladas pelas pessoas, como Inglês, Espanhol e Francês. Elas não foram projetadas por pessoas (embora pessoas tentem impor certa ordem nelas); elas evoluiram naturalmente.

Linguagens formais são linguagens projetadas por pessoas para aplicações específicas. Por exemplo, a notação que os matemáticos usam é uma linguagem formal que é particularmente boa para denotar relacionamentos entre números e símbolos. Químicos usam uma linguagem formal para representar a estrutura química de moléculas. E mais importante:

Linguagens de programação são linguagens formais que foram definidas para expressar computações.

Como mencionei antes, linguagens formais tendem a ter regras estritas sobre sintaxe. Por exemplo, 3+3=6 é uma declaração matemática sintaticamente correta, mas 3 = +6$ não é, Também H2O é um nome químico sintaticamente correto, mas 2Zz não é.

Regras de sintaxe vêm em dois sabores, pertencentes a tokens e estrutura. Tokens são os elementos básicos da linguagem, como palavras e números e elementos químicos. Um dos problemas com 3=+6$ é que o $ não é um token legal na matemática (pelo menos até onde eu sei). Similarmente, 2Zz não é legal porque não existe elemento com a abreviação Zz.

O segundo tipo de erro de sintaxe pertence à estrutura de uma declaração, ou seja, a forma como os tokens são organizados. A declaração 3=+6$ é estruturalmente ilegal, porque você não pode ter um sinal de adição imediatamente depois de um sinal de igualdade. De forma semelhante, fórmulas moleculares tem que ter subscritos depois do nome do elemento, não antes.

Quando lê uma sentença em Inglês ou uma declaração em uma linguagem formal, você tem que descobrir qual é a estrutura da sentença (embora em uma linguagem natural você o faça inconscientemente). Este processo é chamado "parsing". Por exemplo, quando você ouve a sentença, "O outro sapato caiu," você fez o "parse" da sentença. Assumindo que você saiba o que é um sapat, e o que significa cair, você entenderá a implicação geral da sentença.

Embora linguagens formais e naturais tenha muitas características em comum - tokens, estrutura, sintaxe e semântica - existem muitas diferenças.

ambiguidade: Linguagens naturais são cheias de ambiguidades, com as quais as pessoas lidam usando pistas contextuais e outras informações. Linguagens formais são projetadas para ser aproximadamente ou completamente não ambíguas, o que quer dizer que qualquer declaração tem exatamente um significado, independente do contexto.

redundância: Para compensar a ambiguidade e reduzir mal-entendidos, linguagens naturais empregam um monte de redundâncias. Como resultado, elas são frequentemente verbosas. Linguagens formais são menos redundantes e mais concisas.

literalidade: Linguagens naturais são cheias de expressões idiomáticas e metáforas. Se eu disser, "O outro sapato caiu," provavelmente não existe um sapato ou nada caindo. Liguagens formais significam exatamente o que dizem. Pessoas que crescem falando uma linguagem natural (todos) frequentemente têm dificuldade em ajustar-se a linguagens formais. De certa forma a diferença entre linguagem formal natural é como a diferença entre posia e prosa, mas muito mais:

Poesia: Palavras são usadas pelos seus sons assim como pelos seus significados, e o poema inteiro cria um efeito ou resposta emocional. Ambiguidade não é apenas comum mas frequentemente deliberada.

Prosa: O sentido literal das palavras é mais importante e a estrutura contribui mais para para o significado. Prosa é mais passível de análise do que poesia, mas ainda frequentemente ambigua.

Programas: O significado de um programa de computador é não ambíguo e literal, e pode ser entendido inteiramente pela análise dos tokens e estrutura.

Aqui vão algumas sugestões para ler programas (e outras linguagens formais). Primeiro, lembre-se de que linguagens formais são muito mais densas que liguagens naturais, então leva mais tempo para ler. Também, a estrutura é muito importante, então geralmente não é uma boa idéia ler de cima para baixo, da esquerda para a direita. A invés disso, aprenda a fazer o "parse" do programa em sua mente, identificando os tokens e interpretando a estrutura. Finalmente, lembre-se de que os detalhes importam. Pequenas coisas como erros de digitação e pontuação incorreta, das quais você escapa em linguagens naturais, podem fazer uma grande diferença em uma linguagem formal.

1.5. O primeiro programa

Tradicionalmente, o primeiro programa que as pessoas escrevem em um nova linguagem é chamado "Alô, Mundo" porque tudo o que ele faz é escrever as palavras "Alô, Mundo". Em C++ este programa parece com isto:

```
#include <iostream.h>
// main: gera uma simples saída
void main ()
{
    cout << "Alô, Mundo." << endl;
    return 0
}
```

Algumas pessoas julgam a qualidade da linguagem de programação pela simplicidade de seu programa "Alô, Mundo". Por este critério, C++ se sai razoavelmente bem. Mesmo assim, este simples programa contém algumas características que são difíceis de explicar para programadores iniciantes. Por ora, vamos ignorar algumas destas características, como a primeira linha.

A segunda linha começa com //, o que indica que é um comentário. Um comentário é um pedaço de texto escrito em Português (ou em outra língua natual) que você coloca no meio de um programa, normalmente para explicar o que o programa faz. Quando o compilador vê um //, ele ignora tudo daquele ponto até o final da linha.

Na terceira linha, você pode ignorar a palavra void por enquanto, mas observe a palavra main. main é um nome especial que indica o local no programa onde a execução começa. Quando o programa é executado, ele começa executando a primeira declaração int main e continua, na ordem, até atingir a última declaração e então sai.

Não existe limite para o número de declarações que podem estar na função main, mas o exemplo contém apenas um. É uma declaração básica de saída, significando que mostra uma mensagem na tela.
O "cout" é um objeto especial proporcionado pelo sistema que o permite enviar saída para a tela. O símbolo << é um

operador que você aplica a cout e uma string, e que causa a exibição da string.
O "endl" é um símbolo especial que representa o final de uma linha. Quando você envia um endl a cout, ele faz com que o cursor mova para a próxima linha da tela. A próxima vez em que você mandar imprimir alguma coisa, o texto novo aparecerá na próxima linha. Como todas as declarações, a declaração de saída termina com um ponto-e-vírgula (;).
Existem algumas outras coisas que você deve observar sobre a sintaxe deste programa. Primeiro, C++ usa chaves ({ e }) para agrupar coisas. Neste caso, a declaração de saída está entre chaves, indicando que está dentro da definição de main. Note ainda que a declaração está indentada, o que ajuda a mostrar visualmente quais linhas estão dentro da definição.
Neste ponto seria uma boa idéia sentar na frente de um computador e compilar e executar este programa. Os detalhes sobre como fazer isto, dependem do seu ambiente de desenvolvimento, mas deste ponto em diante neste livro, assumirei que você sabe como fazê-lo.
Como mencionei, o compilador C++ é um muito rigoroso com a sintaxe. Se você cometer quaisquer erros quando digitar o programa, as chances são de não compilar com sucesso. Por exemplo, se você digitou incorretamente iostream, você deve obter uma mensagem de erro como a seguinte:

hello.cpp:1: oistream.h: No such file or directory

Existe um monte de informação nesta linha, mas é apresentada em um formato denso que não é fácil de interpretar. Um compilador mais amigável poderia dizer alguma coisa como:

"Na linha 1 do código fonte chamado hello.cpp, você tentou incluir um arquivo header chamado oistream.h. Eu não encontrei nada com este nome, mas encontrei um arquivo chamado iostream.h. Por acaso seria isto o que você quis dizer?"

Infelizmente, poucos compiladores são tão obsequiosos. O compilador não é na verdade tão esperto, e na maioria dos casos as mensagens de erro que você receberá serão apenas uma pista sobre o que está errado. Levará algum tempo para ganhar facilidade em interpretar mensagens do compilador.

No entanto, o compilador pode ser uma ferramenta útil para aprender regras de sintaxe de uma linguagem Começando com um programa funcional (como hello.cpp), modifique-o de várias formas e veja o que acontece. Se você obtiver uma mensagem de erro, tente lembrar-se do que diz a mensagem e o que causou, então se você a ver novamente no futuro, saberá o que significa.

Glossário

linguagem de baixo nível: Uma linguagem de programação que é projetada para ser fácil para o computador executar. Também chamada "linguagem de máquina" ou "linguagem assembly"

portabilidade: Uma propriedade de um programa que pode ser executado em mais de um tipo de computador.

linguagem formal: Qualquer uma das linguagens que as pessoas projetaram para propósitos específicos, como representação de idéias matemática ou programas de computador. Todas as linguagens de programação são linguagens formais.

linguagem natural: Qualquer um das línguas que as pessoas falam que evoluiram naturalmente.

interpretar: A tradução de um linha de cada vez, usada para executar um programa escrito em uma linguagem de alto nível.

compilar: Traduzir um programa em uma linguagem de alto nível para uma linguagem de baixo nível, todo de uma vez, preparando-o para execução futura.
source code: A program in a high-level language, before being compiled.

Código objeto: A saída do compilador, depois da tradução do programa.

executável: Outro nome para o código objeto que está pronto para ser executado.

algorítmo: Um processo geral de solução para uma categoria de problemas.

bug: Um erro em um programa.

sintaxe: A estrutura de um programa.

semântica: O significado de um programa.

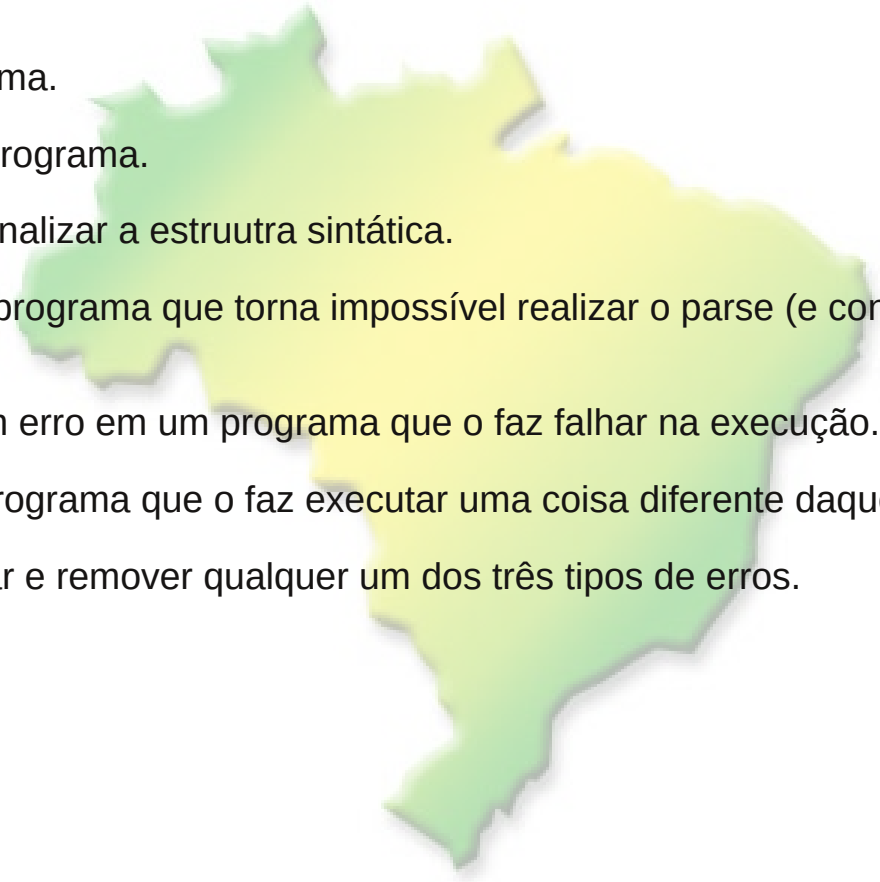
parse: Examinar um program e analizar a estruutra sintática.

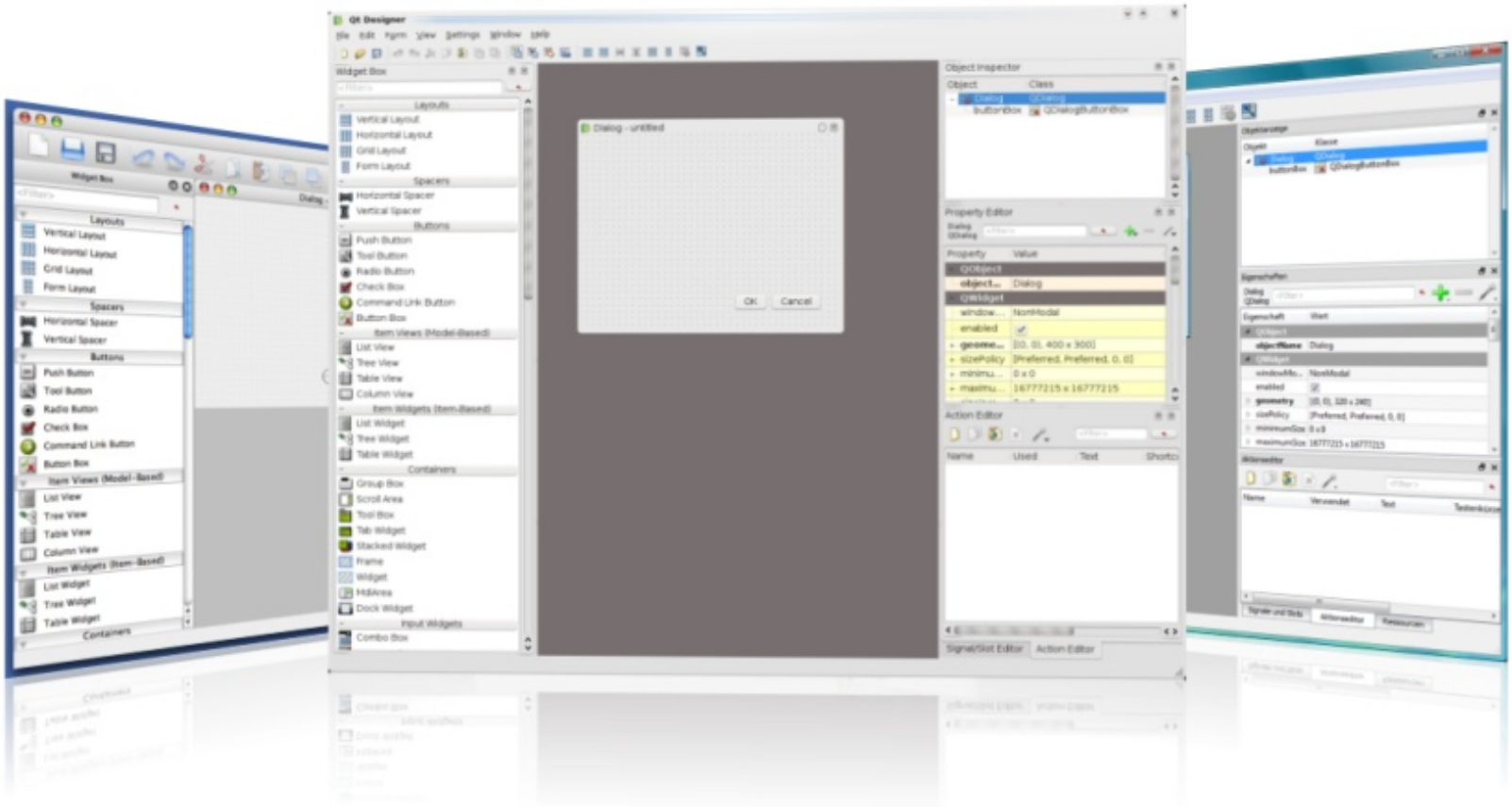
erro de sintaxe: Um erro em um programa que torna impossível realizar o parse (e consequentemente impossível de compilar).

erro em tempo de execução: Um erro em um programa que o faz falhar na execução.

erro de lógica: Um erro em um programa que o faz executar uma coisa diferente daquela pretendida pelo programador.

debugar: O processo de encontrar e remover qualquer um dos três tipos de erros.




Qt Designer
File Edit Form View Settings Window Help
Widget Box
Layouts
Vertical Layout
Horizontal Layout
Grid Layout
Form Layout
Spacers
Horizontal Spacer
Vertical Spacer
Buttons
Push Button
Tool Button
Radio Button
Check Box
Command Link Button
Button Box
Item Views (Model-Based)
List View
Tree View
Table View
Column View
Item Widgets (Item-Based)
List Widget
Tree Widget
Table Widget
Containers
Group Box
Scroll Area
Tool Box
Tab Widget
Stacked Widget
Frame
Widget
MdiArea
Dock Widget
Input Widgets
Combo Box
Dialog - untitled
OK
Cancel
Object Inspector
Object
Class
Property Editor
Property
Value
Action Editor
Name
Used
Text
Signal/Slot Editor
Action Editor