Tutorial de Windows API em C++ usando Paradigma de Dialética Artificial e DSL

oZumbiAnalitico

Todos os prompts foram testados no Grok ou no Copilot da Microsoft. Porém no momento que o texto foi escrito o Grok apresentou melhores resultados e o Copilot da microsoft se mostrou um pouco preguiçoso. É recomendado que seja utilizado os templates prontos sugeridos pela I.A. em vez de configurar projetos vazios, pois existem configurações que mesmo com o auxílio da I.A. não é possível resolver (muito provavelmente porque a A.I. não tem consciência de alguma configuração ou padronização recente). Sempre peça para a I.A. não mostrar os códigos e aguardar instruções, isso vai permitir que ele explique a lógica da estrutura do programa primeiro. Você pode fazer testes iniciais para verificar erros de linkagem, depois basta pedir a implementação de cada arquivo em separado. Não desista na primeira tentativa, programar em linguagens de nível intermediário sempre teve problemas de linkagem, de inserção de bibliotecas e configurações, é o desafio que sobrou para o ser humano, fazer tudo funcionar (e até mesmo nisso, você pode pedir auxílio para a I.A.).

Este é um tutorial para introdução da biblioteca Windows API para utilizar recursos do sistema Windows usando C++. Estaremos utilizando uma abordagem de estudo e desenvolvimento adaptada para o uso de Inteligências Artificiais no auxílio de tarefas como estudo e desenvolvimento de software. A metodologia se baseia nos conceitos de Dialética Artificial e Paradigma DSL. A Dialética Artificial consiste em organizar de maneira estruturada as perguntas necessárias para consciência intelectual de um assunto e usar inteligências artificias para respondê-las. Enquanto o paradigma DSL consiste em utilizar uma pseudo-linguagem (Declarative Semi-Formal Language) para construir a lógica dos programas e instruir a Inteligência Artificial a construir os códigos. Portanto, tanto o estudo quanto a aplicação estará sendo feita sob assistência de Inteligências Artificiais.

Conceitos: Software, Programação, Linguagem de Programação, Código Fonte, Compilador, Interpretador, Biblioteca Padrão, Biblioteca Externa, API.

Um dos conceitos associados ao Paradigma de Dialética Artificial é que você não deve escrever textos explicando tudo que será usado, mas que o leitor busque as definições e as explicações dos conceitos usando inteligências artificiais. O único dever do autor é explicitar os conceitos que estarão sendo utilizados. Fazendo uma analogia a programação, os conceitos são como uma declaração de um arquivo de cabeçalho, enquanto o texto é o código fonte de um arquivo.

Conceitos : Sócrates, Método Socrático, Maiêutica, Ironia Socrática.

```
Dialética [ Metodologia de Estudo e Desenvolvimento ]
* O que é Dialética Artificial ?
** O que é Dialética Socrática ?
*** O que é Estrutura Hierárquica ?
* O que é Paradigma DSL ?
```

[**Dialética Socrática**] no contexto de Dialética Artificial, uma dialética socrática é uma estrutura de perguntas organizadas de maneira hierárquica. A ordem hierárquica está associado ao propósito, uma pergunta tem como propósito a consciência de algum conceito ou a resolução de algum problema, uma subpergunta tem como propósito a consciência de como responder a pergunta a qual ela está subordinada.

Conceitos : Linguagem de Máquina, Linguagem de Baixo Nível, Linguagem de Alto Nível, Linguagem de Nível Intermediário.

O paradigma de DSL busca essencialmente substituir a programação direta em uma linguagem de programação por algo que se situe entre a linguagem humana e uma linguagem de programação de alto nível. Essa linguagem deve fazer uso conveniente de símbolos para instruções definidas que serão interpretadas pela inteligência artificial ao mesmo tempo que deve conter diretivas em linguagem natural que serão interpretadas fazendo uso do potencial intelectual das inteligências artificiais.

Hello World

```
[ Hello World App ] { Windows API, C++ }
1. Inicializar ~ Declarações | Main () || Criar Janela Principal () | & Laço de Mensagens
-> Inicializar ~ Declarações | Main () | Window Procedure () || % WM_PAINT ||
Desenho da Janela || Exibir Texto Hello World
-> Inicializar ~ Declarações | Main () | Window Procedure () || % WM_PAINT |
% WM_DESTROY || Fecha a Janela Corretamente
```

Este é um simples Hello World do windows API. Para você fazer a inteligência artificial ter consciência do paradigma DSL use o arquivo pdf no site (https://github.com/EYO-07/DSL). Depois disso é só copiar e colar a lógica DSL acima, pedindo para Inteligência Artificial usar o pdf como contexto e mostrar o passo a passo da implementação. Caso a I.A. se recuse a ler o pdf, então você pode usar as instruções presentes no arquivo pdf ou na seção **Global Hotkeys App** mais a frente no texto.

A estrutura de uma lógica DSL é organizada em uma identificação do problema (entre colchetes), um contexto (entre chaves) e uma sequência de diretivas que chamo de caminhos. Na notação acima um caminho começa com um número ou com uma seta para que seja possível quebra de linhas.

Conceitos :

1. Variável, Função, Laço, Condicional, Escopo de Função.
2. Sintaxe, Sintaxe de: Variável, Função, Condicional, Switch. Em C++.
3. Tipagem Estática, Tipagem Dinâmica, Tipagem Forte, Tipagem Fraca.
4. Main, Windows Procedure, Forward Declaration, Message Loop, Message.

A linguagem DSL busca aproveitar o espaço ao máximo para compactar as instruções. Por isso os símbolos | e || que são usados para explicitar estruturação hierárquica na mesma linha. $A||B$ significa que B pertence a estrutura de A, ou está subordinado a A, ou pertencem ao mesmo escopo de A, enquanto $A|B$ significa que A e B pertencem a uma mesma estrutura e estão no mesmo nível, sendo B executado depois de A. Os símbolos %, &, () são respectivamente para denotar condicional, laço e função, enquanto os outros símbolos como a enumeração, as setas e $\sim$ tem como finalidade anotação ou clareza.

```
Dialética [ Windows API ]
* O que é Windows API ?
** O que é uma biblioteca em programação ?
** O que é uma API ?
** Qual a diferença entre uma biblioteca e um API ?
* Qual a Finalidade de uma Função Window Procedure ?
** O que é uma mensagem em Windows API ?
** Como funciona a lógica de mensagens em Windows API ?
** Para que serve o Forward Declaration ?
* Qual a Finalidade do Laço de Mensagens ?
* O que é possível fazer com o Windows API ?
```

Não avance muito o texto sem ter consciência das dialéticas e dos conceitos. Faz parte do paradigma de estudo da Dialética Artificial a interação constante com a Inteligência Artificial, este texto (e possivelmente todos os textos a partir desta era) é apenas um catalizador, um roteiro para uma interação mais dinâmica de aprendizado.

```
Dialética [ Mensagens ]{ Windows API }
* Quais mensagens mais comuns em programas usando o Windows API ?
** Utilização || WM_CREATE, WM_PAINT ?
** Utilização || WM_DESTROY, WM_QUIT, WM_CLOSE ?
** Utilização || WM_COMMAND ?
** Utilização || WM_KEYDOWN, WM_KEYUP ?
** Utilização || WM_LBUTTONDOWN, WM_RBUTTONDOWN e WM_MOUSEMOVE ?
** Utilização || WM_SIZE ?
** Utilização || WM_TIMER ?
** Utilização || WM_NOTIFY ?
```

Agora vamos propor uma série de programas simples de forma a varrer de maneira abrangente os recursos do Windows API. O primeiro programa que vamos fazer vai ser um programa para criar atalhos globais de teclado. Como sabemos o windows foi pensado para o usuário comum, porém um programador e um acadêmico está mais acostumado a lidar com o teclado, criar atalhos customizados pode ser extremamente útil. Como queremos atalhos globais que irão servir tanto para a nossa aplicação quanto para qualquer aplicação ativa o programa não irá a princípio utilizar a lógica do Window Procedure. Uma das primeiras coisas que é interessante de fazer é pedir diretamente para a Inteligência Artificial criar um exemplo simples do programa que a gente deseja.

Global Hotkeys App

Conceitos : Paradigma de Modularidade, Módulos, Linker, Diretivas de Preprocessador, Diretiva Include, Diretiva Define, Entry Point.

```
[ Global Hotkeys ] { Windows API, C++ }
1. /functions.cpp || MoveWindow(hwnd, dx, dy) || Move the Window
2. /functions.h || Includes | Function Declarations
3. /main.cpp || Include for functions.h | main() || Register Hotkeys () | & Message Loop |
Unregister Hotkeys ()
-> /main.cpp || Include for functions.h | main() || Register Hotkeys () | & Message Loop ||
Process Hotkeys ()
4. Process Hotkeys () || update the currentHwnd with active window | % currentHwnd ||
% hotkey up, hotkey down, hotkey left, hotkey right || MoveWindow()
5. Register Hotkeys () || Register Win+Alt+UP | Register Win+Alt+DOWN |
Register Win+Alt+LEFT | Register Win+Alt+RIGHT
-> Register Hotkeys () || Register Win+Alt+UP || Register Using MoveWindow
-> Register Hotkeys () || Register Win+Alt+UP | Register Win+Alt+DOWN || ?
-> Register Hotkeys () || ... | Register Win+Alt+LEFT || ?
-> Register Hotkeys () || ... | Register Win+Alt+RIGHT || ?
6. Unregister Hotkeys () || ?
```

Na lógica DSL podemos ter uma ideia de como é a lógica do ponto de entrada do programa pela diretiva (3), o main.cpp é será o arquivo ponto de entrada, dentro dele vamos ter o arquivo organizado em diretivas de inclusão de módulos seguido da função main. Na função main iremos primeiro registrar as hotkeys, depois dentro do laço das mensagens iremos capturar o handle da janela em foco e processar os comandos de atalho do teclado. Use o texto abaixo para explicar para inteligência artificial como interpretar as diretivas da lógica do programa.

```
This is a semiformal declarative language, although there is symbols the
names and descriptions must be interpreted and adapted to create a functional application.
1. N. with N as a number, or ->, don't have any meaning, this is just an human annotation.
2. [ Name or Application Description ]
3. { Context of Application }
4. X ~ Y, means that X is an event and Y is a declarative expression of how the event
should be processed.
5. X || Y means that Y is inside the structure of X, is in his scope.
6. X | Y means that X and Y stands in the same scope (structure) and Y is performed after X.
7. & X, means that X is an repetition structure.
8. X (), denote that X is a function or a task.
9. % X, denote that X belongs to an conditional structure (if-elseif-else).
10. $ X, denotes that X is a declaration for a construction of a variable.
11. X {}, denotes that X is a class structure
12. ... , used to hide parts from the declarative statement which can be deduced from the
previous statements.
13. /filename, denotes a file or a module, /filename || X means that X should be on the
appropriate file.
14. ?, means that should be completed by the A.I.
```

```
Explains step-by-step the creation of the application.
Don't show the code for now, just tell me how to configure the visual studio and create
the suggested files. Wait for further instructions.
```

O texto acima é para ser fornecido para inteligência artificial junto com as diretivas DSL (use shift+enter para pular linhas sem enviar a mensagem para o chatbot). A inteligência artificial vai explicar como configurar o ambiente, os arquivos no visual studio e aguardar interação do usuário. A partir daí basta pedir pela implementação dos arquivos functions.h, functions.cpp e main.cpp. Com tudo pronto vá para o visual studio, compile e teste o programa.

As instruções e as implementações variam um pouco entre as inteligências artificias, mas se tudo der certo o seu programa vai conseguir agora movimentar a qualquer janela em foco usando os atalhos Win+Alt+Seta. Como exercício adicione mais funcionalidades, por exemplo adicione funcionalidade para aumentar ou diminuir o tamanho da janela, para fazer redimensionamento da janela como vertical ou horizontal usando a razão 16:9, funcionalidade para abrir programas que você sempre usa, ou até mesmo executar comandos cmd. Você pode fazer isso usando diretivas DSL ou programando diretamente e usando o código gerado pelas instruções iniciais como template.

```
[ Paradigma de Dialética Artificial e DSL ]
1. Estudo ~ & Texto || % Dúvidas || Montar Dialética | & Estudar Dialética ||
& Perguntas | & Exemplos
-> Estudo ~ & Texto || % Dúvidas | % Dialéticas || & Estudar Dialética
-> Estudo ~ & Texto || % Dúvidas | % Dialéticas | % Exemplos ou Exercícios ||
Tentar Aplicar os Exemplos ou Exercícios | % Dúvidas || Adicionar a Dialética
-> ... || & Perguntas || Tentar Formular a Resposta | Perguntar para I.A. |
% Dúvidas || Adicionar a Dialética
-> ... | & Exemplos || % Dúvidas || Adicionar a Dialética
```

A lógica acima mostra também que o DSL pode ser usado com finalidades mais gerais, como estabelecer métodos, workflows e etc.

Simple Notepad

Com a finalidade de aprender a criar interfaces gráficas usando o Windows API vamos usar como programa base um notepad. Este notepad vai ter um menu com botões de salvar, carregar arquivo, abas, usar caixas de diálogos comuns, e uma opção de configurações.

```
[ Interface Gráfica ]
* Quais recursos de interface gráfica temos no Windows API ?
** Exemplo de sintaxe para criação de Windows em Windows API ?
** Exemplo de sintaxe para criação de Menus em Windows API ?
** Exemplo de sintaxe para criação de Caixas de Diálogo em Windows API ?
** Exemplo de sintaxe para criação de Controles em Windows API ?
* Quais recursos de interface gráfica a biblioteca de controle comum fornece ?
```

Conceitos :

1. Interface de Linha de Comandos, Interface Gráfica.
2. Window, Menu, Caixas de Diálogo, Controles, Controle de Edição, Botões, Caixas de Seleção, Listas, Barras de Rolagem.
3. Barras de Ferramentas, Barras de Status, Barras de Progresso, Controle de Abas, Listas de Visualização, Spin Control, Trackbars, Tree Controls, Caixas de Listas e Caixas de Lista Combinadas.
4. Declaração de Função, Definição de Função, Chamada de Função.

Antes de fornecer as instruções para I.A., será preciso novamente mencionar a necessidade de ter consciência dos possíveis erros que podem acontecer durante a construção do programa. Já é um exercício fazer funcionar os códigos fornecidos pela inteligência artificial, pois mesmo seguindo os passos da I.A. corretamente ainda haverá a necessidade de fazer pequenos ajustes ou mesmo modificações na lógica.

Conceitos : Erros de Sintaxe, Erros de Segmentação, Vazamento de Memória, Erros de Ponteiro, Erros de Tipo, Overflow, Underflow, Erro de Escopo de Variáveis, Erros de Compilação, Erros de Linkagem, Erros de Gerenciamento de Biblioteca.

Quando houver erros use a intuição, não siga os conselhos da I.A. cegamente, pois ela não tem consciência de todas as variáveis e tem sua consciência limitada ao seu período de treinamento. Não considero improvável que ela termine sugerindo reinstalar o sistema operacional só porque você não soube explicar corretamente o que estaria acontecendo.

[Simple Notepad] { Windows API, C++, x64 }
1. /global_vars.h | $ int índice da aba corrente | $ int contador de abas |
$ HWND Controle de Abas | $ HWND Botão para Adicionar novas Abas |
$ Vetor de HWND de edit controls associado as abas |
$ Vetor de wstrings para adicionar nomes de arquivos associado as abas
2. /main.cpp || Includes | Declarações | Main () | Definições de Procedures
-> ... | Definições de Procedures || Window Procedure da Janela Principal () |
Window Procedure do Edit Control ()
-> ... Main () || Criar Janela Principal | & Laço de Mensagens
3. WM_CREATE da Janela Principal ~ Cria um Menu | Cria o HWND Controle de Abas |
Adiciona uma Aba Inicial | Adiciona um Edit Control no Vetor de Edit Controls
-> WM_CREATE da Janela Principal ~ Cria um Menu || Adiciona o Botão Load |
Adiciona o Botão Save
4. Ativação do Botão Load do Menu ~ Load ()* || Abre Diálogo para Carregar Arquivo |
% arquivo selecionado ||
Adiciona no Vetor de nomes de Arquivos o nome do Arquivo com o índice da aba corrente |
% existe um edit control associado ao índice da aba corrente ||
Carrega o Conteúdo do Arquivo no Edit Control
-> ... | % existe um edit control associado ao índice da aba corrente | % else ||
Cria um Edit Control no vetor de Edit Controls associado a Aba Corrente
-> Ativação do Botão Load do Menu ~ Load ()* || Abre Diálogo para Carregar Arquivo |
% arquivo selecionado | % Erro || Mostra MessageBox de Mensagem de Erro
5. Ativação do Botão Save do Menu ~ Save ()*
6. Window Procedure do Edit Control () || % Combinação de Teclas Ctrl+S || Save ()*
-> Window Procedure do Edit Control () || % Combinação de Teclas Ctrl+S |
% Combinação de Teclas Ctrl+T || Adiciona Nova Aba | Aumenta uma unidade do contador de abas
7. WM_CLOSE da Janela Principal ~ Abre um Diálogo Perguntando se Quer Fechar o Programa |
Fecha o Programa ou não de Acordo com o Resultado do Diálogo
8. /functions.cpp ~ ... | Load ()? | Save ()? ||
% temos um edit control associado ao índice da aba corrente ||
% temos um nome arquivo com o índice da aba corrente ||
Salva o conteúdo do edit control no arquivo especificado
-> /functions.cpp ~ ... | Load ()? | Save ()? ||
% temos um edit control associado ao índice da aba corrente ||
% temos um nome arquivo com o índice da aba corrente | % else ||
Abre um diálogo para salvar arquivo | % temos um nome de arquivo ||
Cria um arquivo Vazio com o nome e Adiciona no Vetor de nomes de arquivos com o índice
da aba correspondente
-> /functions.cpp ~ ... | Load ()? | Save ()? ||
% temos um edit control associado ao índice da aba corrente | % else ||
Adiciona um Edit Control Vazio e adiciona no vetor | Abre um diálogo para salvar arquivo |
% temos um nome de arquivo ||
Cria um arquivo Vazio com o nome e Adiciona no Vetor de nomes de arquivos com o índice
da aba correspondente
-> /functions.h ~ ... | Save ()! | Load () ! | Mostra a Aba Ativa ()! | ...
-> /functions.cpp ~ ... | Load ()? | Save ()? | Mostra a Aba Ativa ()? ||
atualiza o índice da aba corrente com a aba ativa |
Deixa Visível somente o Edit Control do Vetor que está associado a aba ativa
9. Notificação de Mudança nas Abas ~ Mostra a Aba Ativa ()*

Como foram adicionados novas instruções do DSL adicione as instruções abaixo no prompt.

```
This is a semiformal declarative language, although there is symbols the names and
descriptions must be interpreted and adapted to create a functional application.
1. N. with N as a number, or ->, don't have any meaning, this is just an human annotation.
2. [ Name or Application Description ]
3. { Context of Application }
4. X ~ Y, means that X is an event or task and Y is a declarative expression of
how the event should be processed.
5. X || Y means that Y is inside the structure of X, is in his scope.
6. X | Y means that X and Y stands in the same scope (structure) and Y is
performed after X.
7. & X, means that X is an repetition structure.
8. X (), denote that X is a function or a task. If it is an actual function
depends of implementation.
9. X ()!, denote that it's a function declaration.
10. X ()?, denote that it's a function definition.
11. X ()*, denote that it's a function call.
12. % X, denote that X belongs to an conditional structure (if-elseif-else).
13. $ X, denotes that X is a declaration for a construction of a variable.
14. X {}, denotes that X is a class structure
15. ... , used to hide parts from the declarative statement which can be deduced from the
previous statements.
16. /filename, denotes a file or a module, /filename || X means that X should be on the
appropriate file.
17. ?, means that should be completed by the A.I.
Explains step-by-step the creation of the application.
Don't show the code for now, just tell me how to configure the visual studio and create
the suggested files. Wait for further instructions.
```