西川-本のテスト

aiueo

+ aiueo

- New Page
- Doxygenで生成
- Doxygenでクラス図の表示
- 文字コード概要
- DrawTextとDrawTextLayoutの違い

New Page

aiueo

- aiueo

1. aioue
2. huid

aiueo

aejo

a

> aoiuehpihps '

`s`

Doxygenで生成

> ソースコードから自動的にドキュメントを生成するツール。

- C++、C言語、Java、C#など

⚠下記のzennのサイトを参考にすべし。

DoxygenGUI frontendでグラフィカルに設定できるよ

就活には自動生成ドキュメントが効くらしい

設定ファイルを生成

```
doxygen -g
```

カレントディレクトリに、Doxyfileという名前で出力される。

設定ファイルを編集

Doxygen + Graphviz を使って呼び出し依存関係図を生成する - Posts - Seaside Laboratory

- テキストエディタでDoxyfile編集をする。

項目名	意味	設定例
PROJECT_NAME	プロジェクト名	"Knuckle Fighter Maniax"
OUTPUT_DIRECTORY	ドキュメントの出力先	"C:\My Program\doxygen"
OUTPUT_LANGUAGE	出力言語	Japanese
FULL_PATH_NAMES	出力ファイル名をフルパスにする	NO
OPTIMIZE_OUTPUT_FOR_C	C 言語に最適化	YES
EXTRACT_ALL	全てを展開	YES
INPUT	ソースファイルのパス	"C:\My Program"
INPUT_ENCODING	ソースファイルの文字コード	SHIFT_JIS
FILE_PATTERNS	対象とするファイル	*.cpp *.h
RECURSIVE	サブディレクトリーも含める	YES
VERBATIM_HEADERS	ドキュメントにヘッダーを引用	NO
GENERATE_TREEVIEW	ツリービューの利用	YES
ENUM_VALUES_PER_LINE	列挙型を行に並べる数	1
GENERATE_LATEX	LATEX 版の出力	NO

ドキュメント作成

- 設定ファイルのあるディレクトリで以下のコマンドを実行。

```
doxygen
```

- 出力先に「html」という名前のフォルダが自動で作成されるので、その中にあるindex.htmlを開く。

⚠
//Graphivizも使った方法はのページに

Doxygenでクラス図の表示

【Windows】Doxygenをインストールして使う | The modern stone age. 上記を参考に。

↓これを出力先とかいい感じに変えて使って

Doxyfileの設定

項目名	意味	設定例
PROJECT_NAME	プロジェクト名	"Knuckle Fighter Maniax"
OUTPUT_DIRECTORY	ドキュメントの出力先	"C:\My Program\doxygen"
OUTPUT_LANGUAGE	出力言語	Japanese
FULL_PATH_NAMES	出力ファイル名をフルパスにする	NO
OPTIMIZE_OUTPUT_FOR_C	C 言語に最適化	YES
EXTRACT_ALL	全てを展開	YES
INPUT	ソースファイルのパス	"C:\My Program"
INPUT_ENCODING	ソースファイルの文字コード	SHIFT_JIS
FILE_PATTERNS	対象とするファイル	*.cpp *.h
RECURSIVE	サブディレクトリーも含める	YES
VERBATIM_HEADERS	ドキュメントにヘッダーを引用	NO
GENERATE_TREEVIEW	ツリービューの利用	YES
ENUM_VALUES_PER_LINE	列挙型を行に並べる数	1
GENERATE_LATEX	LATEX 版の出力	NO

Graphviz関係の設定

項目名	意味	設定例
HAVE_DOT	dot ツールの有無	YES
DOT_NUM_THREADS	グラフ生成時のスレッド数	4
UML_LOOK	継承図を UML 風にする	YES
CALL_GRAPH	呼び出し依存関係図の生成	YES
CALLER_GRAPH	呼び出し元依存関係図の生成	YES
DOT_PATH	dot ツールの位置	"C:\Program Files\Graphviz\bin”
設定項目	説明	
---	---	
OUTPUT_DIRECTORY	ドキュメントの出力ディレクトリ。	
OUTPUT_LANGUAGE	ドキュメントの出力言語。	
OPTIMIZE_OUTPUT_JAVA	Java向けに出力を最適化するかどうか。	
EXTRACT_ALL	全てのメンバーを抽出するかどうか。	
EXTRACT_PRIVATE	プライベートメンバーを抽出するかどうか。	
EXTRACT_PRIV_VIRTUAL	プライベートな仮想メンバーを抽出するかどうか。	
EXTRACT_PACKAGE	パッケージメンバーを抽出するかどうか。	
EXTRACT_STATIC	スタティックメンバーを抽出するかどうか。	
EXTRACT_LOCAL_CLASSES	ローカルクラスを抽出するかどうか。	
INPUT	ソースファイルの入力ディレクトリ。	
RECURSIVE	入力ディレクトリを再帰的に検索するかどうか。	
REFERENCED_BY_RELATION	被参照関係を生成するかどうか。	
REFERENCES_RELATION	参照関係を生成するかどうか。	

項目名	意味	設定例
ALPHABETICAL_INDEX	アルファベット順の索引を生成するかどうか。	
GENERATE_LATEX	LaTeX形式のドキュメントを生成するかどうか。	
UML_LOOK	UMLスタイルの図を生成するかどうか。	
UML_LIMIT_NUM_FIELDS	UML図に表示するフィールドの最大数。	
TEMPLATE_RELATIONS	テンプレートの関係を表示するかどうか。	
CALL_GRAPH	呼び出しグラフを生成するかどうか。	
CALLER_GRAPH	呼び出し元グラフを生成するかどうか。	
DOT_IMAGE_FORMAT	DOTグラフの出力フォーマット。	
DOT_GRAPH_MAX_NODES	DOTグラフに表示する最大ノード数。	

文字コード概要

C言語の場合

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

int main()
{
    /* 2つの文字列を結合する：C言語*/
    const char* strA = "Hello, ";
    const char* strB = "world.";

    /* 領域確保 */
    char* strC = (char*)malloc(strlen(strA) + strlen(strB) + 1);

    strcpy(strC, strA); /* strAをstrCにコピー */
    strcat(strC, strB); /* strBをstrCに結合 */
    puts(strC);

    if (strcmp(strC, "Hello, world.") == 0) /* 比較 */
    {
        puts("ok.");
    }
    free(strC); /* 領域解放 */
}
```

C++の場合

```
#include <iostream>
#include <string>

int main()
{
    // 2つの文字列を結合する：C++
    std::string strA = "Hello, ";
    std::string strB = "world.";

    std::string strC = strA + strB; // 結合
    std::cout << strC << std::endl;

    if (strC == "Hello, world.") // 比較
    {
        std::cout << "ok." << std::endl;
    }
    // 確保された領域は自動的に解放される
}
```

- std::stringクラスの実体は`std::basic_string<charT , traits,Allocator>`というテンプレートクラス

文字コード

- 文字は数値、文字列は数値の並びとして扱われる。
- 入出力の際に文字と数字との対応表を文字コードという

昔の文字コード

> ASCIIコード

- 7bit(0〜127 : 0x00〜0x7F)で表現できる数値のそれぞれに文字を割り当てたもの
- 昔は英数字、数字、いくつかの記号、改行やタブなどの制御記号しか扱っていなかった

今流行りの拡張

MBCS : Multi Byte Character Set

- 複数のcharの組に1文字を割り当てる。
  - 半角文字は1バイトだが、全角文字は2バイトとか3バイトとか
- 以前から使ってきた文字列(char*,char[])をそのまま使えるが、char配列の文字列の長さと文字列の長さが一致しなくなる。

C 日本語文字列 - yonewiki

- UTF-8、Shift-JISなど。
- **std::stringはMBCSを扱っている。**

WSC : Wide Character Set

マルチバイト文字とワイド文字

- charより大きなサイズの型「wchar_t」を用意してwchar_tの列で文字列を表す
- **ワイド文字列**とも呼ばれる
- 常に16ビットなので、英数字しか使わない場合はMBCSより文字列を確保するのに必要な領域が増える
- Unicode(UTF-16)
- **std::wstringがWSCを扱っている**

DrawTextとDrawTextLayoutの違い

テキストを描画する方法 - Win32 apps

DrawText

- 簡単なテキスト描画向け
- テキスト全体に対して同じ書式設定を適用する場合に使われる

DrawTextFormat

- 複雑なレイアウトのテキスト描画向け
- テキストとそのレイアウトを一緒に保持するため、テキストの内容、書式が変わらない場合に有効
- テキストを変える際は解放→再生成しないといけない