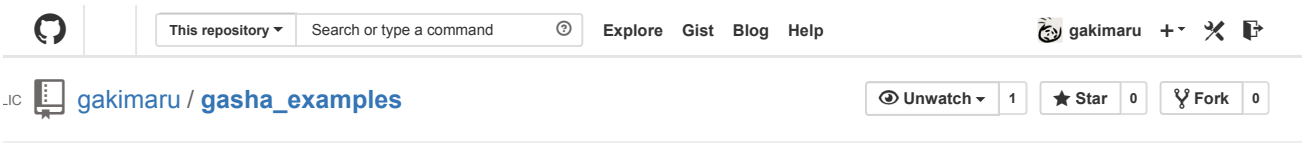

branch: dev/iss/#0003 gasha_examples / README.md

gakimaru 9 hours ago ・全てのマクロ名を調整。（必ず GASHA_ で始まる）

1 contributor

file 636 lines (565 sloc) 42.598 kb Open Edit Raw Blame History Delete

GASHA

Gakimaru's researched and standard library for C++

サンプルプログラム用リポジトリ

このリポジトリは、「GASHA」のサンプルプログラム用です。

> 多数のサンプルプログラムのビルドと実行が可能です。
>
> また、サブモジュールにより、ライブラリ用の各リポジトリを適切な位置関係に配置しており、ライブラリファイルのビルド、および、ソースコードレベルでのライブラリのデバッグが可能です。

関連リポジトリ

- `gasha` ライブラリ本体用リポジトリ
- `gasha_settings` ライブラリ挙動設定用リポジトリ
- `gasha_src` ライブラリソース用リポジトリ
- `gasha_examples` サンプルプログラム用リポジトリ
- `gasha_proj` ライブラリビルド用リポジトリ

「GASHA」とは？

個人制作による、オープンソースのC++用基本ライブラリです。

> 主にゲーム開発での利用を想定し、暗黙的なメモリ確保／解放を行わない点が特徴の一つです。
> 高速性もしくは生産性を追求し、品質と開発効率の向上を目的とするものです。
> C++11を基盤に開発しているため、プラットフォーム／コンパイラ依存のコードが少なく、それによって標準化されたマルチスレッド処理対応が多い点も特徴です。

基本的なアルゴリズム、コンテナ、メモリ管理、デバッグ機能などを用意しています。

> STLライクなコンテナは、データ構造とアルゴリズムを分離した構成としており、ユーザー独自のデータを一時的にコンテナ化して操作するなど、柔軟に活用することができます。
> 他、メタプログラミング向けのテンプレートなども用意しています。
> 個人的な学習・調査・研究に応じて、気まぐれに追加します。

名前の由来

> 名前の由来は「がしゃどくろ」から。
> その「骨格」的な意味合いに、自身のハンドル名「Gakimaru」の頭文字「GA」を重ねて命名しました。

ライブラリサンプルプログラム実行手順

【クイックスタート】

1. `git` で `gasha_examples` リポジトリのクローンを作成

   > $ git clone https://github.com/gakimaru/gasha_examples.git

2. **サブモジュールのアップデート**

   > $ cd gasha_examples
   > $ git submodule update --init
   >
   > これにより、サブモジュールとして配置された三つのリポジトリクローンが更新されます。
   >
   > > `sub/gasha` ライブラリ本体
   > > `sub/gasha_settings` ライブラリ挙動設定
   > > `sub/gasha_src` ライブラリソース

3. **ライブラリのビルド**

   > **【Visual C++の場合】**
   >
   > > `proj/gasha_examples.sln` を開き、`[ビルド]→[バッチビルド]` メニューを実行し、すべての（もしくは任意の）プロジェクト／構成／プラットフォームを選んでビルドを実行して下さい。
   >
   > **【Unix系環境+GCCの場合】**
   >
   > > proj/mk_all.sh を実行して下さい。
   > >
   > > > $ cd proj
   > > > $ ./mk_all.sh all
   > >
   > > ルビルドする場合は、一度 ./mk_all.sh clean を実行してから ./mk_all.sh all を実行して下さい。

4. **サンプルプログラムの実行**

   > **【Visual C++の場合】**
   >
   > > `exe/vc/` 以下のディレクトリから、`（サンプル名）.exe` を実行することで、サンプルプログラムを実行可能です。
   > > `exe/vc/` 以下のディレクトリには、サンプルプログラムをまとめて実行し、結果をログファイルに記録するためのバッチファイルも多数用意しています。
   > > また、Visual Studio 上から、「スタートアッププロジェクト」を設定すれば、デバッグ実行することも可能です。
   >
   > **【Unix系環境+GCCの場合】**
   >
   > > `exe/gcc/` 以下のディレクトリから、`（サンプル名）` の実行ファイルを実行することで、サンプルプログラムを実行可能です。
   > > `exe/gcc/` 以下のディレクトリには、サンプルプログラムをまとめて実行し、結果をログファイルに記録するためのシェルスクリプトも多数用意しています。

対象コンパイラ

ライブラリの利用、および、サンプルプログラムの実行には、**C++11仕様に対応したコンパイラ**が必要です。

【推奨コンパイラ】

- Visual C++ 12.0 (Visual Studio 2013) 以降
- GCC 4.6 以降

開発には下記のコンパイラを用いています。

- Visual C++ 12.0 (Visual Studio 2013)
- GCC 4.8.2 (32bit版Cygwin環境)

> 「可変長引数テンプレート」使用箇所を削除するなど、わずかな改変でC++11以前のコンパイラにも適用できるものが多数あります。
> 逆に言えば、C++11仕様にさえ対応していれば、上記以外のコンパイラでも、ほとんど改変することなく利用可能です。
> Clang での動作は未確認ですが、おそらく適用可能です。

基本ディレクトリ構成とサブモジュール構成

サンプルプログラム＆ライブラリビルド用リポジトリ

```
[gasha_examples]       ... サンプルプログラム用リポジトリ
 |
 |-[exe]               ... 実行ファイル用
 |-[proj]              ... 各サンプルプログラムのビルドプロジェクト用
 |-[src]               ... 各サンプルプログラムのソースファイル用
 `-[sub]               ... サブモジュール用
    |
    |-[gasha]          ... ライブラリ本体用リポジトリ(https://github.com/gakimaru/gasha)
    |-[gasha_settings] ... ライブラリ挙動設定用リポジトリ(https://github.com/gakimaru/gasha_settings)
    `-[gasha_src]      ... ライブラリソース用リポジトリ(https://github.com/gakimaru/gasha_src)
```

ライブラ単体リビルド用リポジトリ

```
[gasha_proj]           ... ライブラリビルド用リポジトリ
 |
 |-[proj]              ... ライブラリビルドプロジェクト用
 `-[sub]               ... サブモジュール用
    |
    |-[gasha]          ... ライブラリ本体用リポジトリ(https://github.com/gakimaru/gasha)
    |-[gasha_settings] ... ライブラリ挙動設定用リポジトリ(https://github.com/gakimaru/gasha_settings)
    `-[gasha_src]      ... ライブラリソース用リポジトリ(https://github.com/gakimaru/gasha_src)
```

ライブラリ概要

環境

- `gasha/build_settings/build_settings.h` **プラットフォーム／言語設定**

> 説明
>
> - コンパイラの種類やバージョンを判別し、必要に応じて `nullptr`, `override`, `alignas`, `thread_local` などのC++11仕様に合わせた処理の独自実装版を有効化し、コード互換性の向上に寄与します。
> - 同様に、`__FUNCTION__`, `__PRETTY_FUNCTION__` なども各コンパイラで共通利用可能にします。
> - 通常このヘッダーファイルは、強制インクルード ファイルに設定して利用します。
>
> 資料
>
> - **本当にちょっとしたプログラミングTips.pdf**
>   ［コーディングに関するTips］-［#defineマクロの活用］

ユーティリティ

- `gasha/utility.h` ［テンプレート］**汎用ユーティリティ** *1
- `gasha/type_traits.h` ［テンプレート］**型特性ユーティリティ** *2

> 説明
>
> - 汎用的なユーティリティ関数／クラスです。
>   メタプログラミングにも対応し、コンパイル時の情報取得に活用できます。
>
> > 注釈
> >
> > - *1：不定長 min() / max() 関数や、値を交換するための swapValues() 関数などを扱います。
> > - *2：配列の要素数を取得するための extentof(), rankof() などを扱います。
>
> 資料
>
> - **効果的なテンプレートテクニック.pdf**
>   ［メタプログラミング］

算術：基本

- `gasha/basic_math.h` ［テンプレート含む］**基本算術**

> 説明
>
> - べき乗、素数判定、素数計算、ビット数／MSB／LSB計算といった基本的な算術系処理を扱います。
> - メタプログラミングに対応し、コンパイル時に計算結果を算出することができます。
>
> 資料
>
> - **効果的なテンプレートテクニック.pdf**
>   ［メタプログラミング］

算術：CRC計算

- `gasha/crc32.h` **CRC32計算**

> 説明
>
> - CRC計算を扱います。現状、32ビットのみに対応しています。
> - C++11の `constexpr`, `ユーザー定義リテラル` による、メタプログラミング版（コンパイル時計算）に対応しています。
>   GCCは、constexprが4.6から、ユーザー定義リテラルが4.7から対応しています。
>   Visual C++ 12.0(VS2013)はどちらも未対応ですが、constexprは次期バージョンで対応予定です。
> - ランタイム時のCRC計算は、高速なSSE命令に対応します。
>   ただし、SSE命令使用時の多項式はCRC-32Cに限定します。
>   SSE命令を使用しないように指定することもできます。
>
> 資料
>
> - **効果的なテンプレートテクニック.pdf**
>   ［メタプログラミング］-［constexprの活用：CRC値の算出］

アルゴリズム：ソート

【交換ソート】

- `gasha/bubble_sort.h` ［テンプレート］**バブルソート** *1
- `gasha/shaker_sort.h` ［テンプレート］**シェーカーソート** *1
- `gasha/odd_even_sort.h` ［テンプレート］**奇遇転置ソート** *1
- `gasha/shear_sort.h` ［テンプレート］**シェアソート**
- `gasha/comb_sort.h` ［テンプレート］**コムソート**
- `gasha/gnome_sort.h` ［テンプレート］**ノームソート** *1

【分割交換ソート】

- `gasha/quick_sort.h` ［テンプレート］**クイックソート**

【選択ソート】

- `gasha/selection_sort.h` ［テンプレート］**選択ソート** *1
- `gasha/heap_sort.h` ［テンプレート］**ヒープソート**

【挿入ソート】

- `gasha/insertion_sort.h` ［テンプレート］**挿入ソート** *1 *3
- `gasha/shell_sort.h` ［テンプレート］**シェルソート** *3

【マージソート】

- `gasha/inplace_merge_sort.h` ［テンプレート］**インプレースマージソート** *1

【混成ソート】

- `gasha/intro_sort.h` ［テンプレート］**イントロソート** *3

【分布ソート】 ※非比較ソート

- `gasha/radix_sort.h` ［テンプレート］**基数ソート** *1 *2

> 説明
>
> - 配列もしくはコンテナのデータをソートします。

> - 通常は「イントロソート」、コードサイズを削減したければ「コムソート」、連結リストなら「挿入ソート」、メモリ度外視で大量データを高速ソートしたければ「分布ソート」、といった使い分けができます。
>
> 注釈
>
> - *1：安定ソートです。
> - *2：外部ソートです。内部で一時的にメモリ確保します。
> - *3：配列以外のコンテナ（イテレータ）にも対応しています。
>
> 資料
>
> - **プログラム最適化Tips.pdf**
> ［データ構造とアルゴリズムによる最適化］-［ソートアルゴリズム］

アルゴリズム：探索

- `gasha/linear_search.h` ［テンプレート］**線形探索** *1
- `gasha/binary_search.h` ［テンプレート］**二分探索** *1 *2

> 説明
>
> - 配列もしくはコンテナのデータを探索します。
>
> 注釈
>
> - *1：配列以外のコンテナ（イテレータ）にも対応しています。
> - *2：ソート済み状態を前提とした探索です。重複キーは必ず先頭にマッチします。
>
> 資料
>
> - **プログラム最適化Tips.pdf**
> ［データ構造とアルゴリズムによる最適化］-［探索アルゴリズム］

マルチスレッド：ロック

- `gasha/spin_lock.h` **スピンロック** *1
- `gasha/lw_spin_lock.h` **サイズ軽量スピンロック** *1 *2
- `gasha/dummy_lock.h` **ダミーロック** *1 *3
- `gasha/lock_guard.h` ［テンプレート］**スコープロック** *4
- `gasha/unique_lock.h` ［テンプレート］**単一ロック** *5

> 説明
>
> - スレッド間のロック制御を行います。
> - 通常、スピンロックはマルチコア環境に最適であり、シングルコア環境ではプリエンプション（タイムスライスを使い切る）までコンテキストスイッチが発生しないため、非効率です。
> - その対策として、一定回数のスピンでコンテキストスイッチを行う仕組みを組み込んでいます。
>
> 注釈
>
> - *1：スコープロックパターン制御（lock_guard, unique_lock）に対応しています。
> - *2：通常版が4バイトに対して、軽量版が1バイトです。通常版の方が若干高速です。
> - *3：ロッククラスと同じインターフェースを備えますが、実際にはロック制御しません。
> コンテナのロック制御を無効化する時などに使用します。
> - *4：スコープロックによるロック制御を行います。コンストラクタでロックを自動的に取得します。
> - *5：スコープロックによるロック制御を行います。コンストラクタでロックを取得するかどうかを選べます。
>
> 資料
>
> - **プログラム最適化Tips.pdf**
> ［データ構造とアルゴリズムによる最適化］-［マルチスレッドを最適化するアルゴリズム］
> - **マルチスレッドプログラミングの基礎.pdf**
> ［様々な同期手法］-［排他制御：スピンロック］

マルチスレッド：共有ロック

- `gasha/shared_spin_lock.h` **共有スピンロック** *1
- `gasha/simple_shared_spin_lock.h` **単純共有スピンロック** *1 *2

- `gasha/unshared_spin_lock.h` **非共有スピンロック** *1 *3
- `gasha/dummy_shared_lock.h` **ダミー共有ロック** *1 *4
- `gasha/shared_lock_guard.h` ［テンプレート］**スコープ共有ロック** *5
- `gasha/unique_shared_lock.h` ［テンプレート］**単一共有ロック** *6

> 説明
>
> - 共有ロック（リードロック）と排他ロック（ライトロック）を使い分けたロック制御を行います。
> - 共有ロックは複数のスレッドが同時に取得できますが、排他ロックは一つのスレッドしか取得できません。
> - 一定回数のスピンでコンテキストスイッチを行います。
>
> > 注釈
> >
> > - *1：スコープロックパターン制御（lock_guard, shared_lock_guard, unique_shared_lock）に対応しています。
> > - *2：排他ロックを優先しません。 通常版は、排他ロック取得待ちが発生すると、後続の共有ロックが取得待ち状態になります。 単純版は、共有ロックが混み合うと、排他ロックが取得できなくなります。
> > - *3：共有ロッククラスと同じインターフェースを備えますが、実際には全て排他ロックします。
> > - *4：共有ロッククラスと同じインターフェースを備えますが、実際にはロック制御しません。
> >   コンテナのロック制御を無効化する時などに使用します。
> > - *5：スコープロックによる共有ロック制御を行います。コンストラクタでロックを自動的に取得します。
> > - *6：スコープロックによる共有ロック制御を行います。コンストラクタでロックを取得するかどうかを選べます。
>
> 資料
>
> - **プログラム最適化Tips.pdf**
>   ［データ構造とアルゴリズムによる最適化］-［マルチスレッドを最適化するアルゴリズム］
> - **マルチスレッドプログラミングの基礎.pdf**
>   ［様々な同期手法］-［排他制御：リード・ライトロック］

マルチスレッド：共有データ

- `gasha/shared_pool_allocator.h` ［テンプレート］**共有プールアロケータ** *1
- `gasha/shared_stack.h` ［テンプレート］**共有スタック** *1
- `gasha/shared_queue.h` ［テンプレート］**共有キュー** *1

> 説明
>
> - マルチスレッドでデータを共有するためのテンプレートクラスです。
> - コンテナとしては機能しません。至ってシンプルなアロケート／フリー／プッシュ／ポップ／エンキュー／デキューのみに対応します。
>
> > 注釈
> >
> > - *1：共有にはロック制御を伴いますが、使用する同期オブジェクトをテンプレート引数で指定します。
> >   デフォルトは spinLock ですが、std::mutex（C++11標準ライブラリ）などに変更することができます。
> >   dummyLock を指定すれば、ロック制御を無効化し、シングルスレッドで高速に動作させることができます。
>
> 資料
>
> - **プログラム最適化Tips.pdf**
>   ［データ構造とアルゴリズムによる最適化］-［マルチスレッドを最適化するアルゴリズム］

マルチスレッド：ロックフリー共有データ

- `gasha/lockfree_pool_allocator.h` ［テンプレート］**ロックフリープールアロケータ**
- `gasha/lockfree_stack.h` ［テンプレート］**ロックフリースタック** *1
- `gasha/lockfree_queue.h` ［テンプレート］**ロックフリーキュー** *1
- `gasha/tagged_ptr.h` ［テンプレート］**タグ付きポインター**

> 説明
>
> - ロックフリーアルゴリズムの実践です。
> - マルチスレッドでデータを共有するためのテンプレートクラスですが、ロック制御をしません。
> - コンテナとしては機能しません。至ってシンプルなアロケート／フリー／プッシュ／ポップ／エンキュー／デキューのみに対応します。
> - 「キャッシュコヒーレンシ問題」、「ABA問題」に対処済みです。

- スレッドが混み合うような状況下では、通常のロック制御よりも効率的に動作します。
- 逆に競合が少ない場面では、スピンロックによるロック制御の方が高速です。
- なお、ロックフリーアルゴリズムは、シングルコア環境でも効果を発揮します。

注釈

- *1：ロックフリーアルゴリズム特有の「ABA問題」に対処するために、「タグ付きポインター」を使用します。
  タグ付きポインターは、ポインターの一部にタグとなる情報を仕込んで、ポインターの誤認を防ぐ手法です。
  32ビット環境では、ポインターを64ビットに拡張して32ビットのタグを扱うことでABA問題をほぼ完全に防ぎます。
  64ビット環境では、アラインメントの隙間（2〜4びっとほど）か、実質的な有効アドレス範囲外の上位8ビットなどを利用します。

資料

- **プログラム最適化Tips.pdf**
  ［データ構造とアルゴリズムによる最適化］-［マルチスレッドを最適化するアルゴリズム］
  ※ABA問題やキャッシュコヒーレンシ問題についても解説しています。

ライブラリの命名規則

ライブラリに用いている命名規則は、基本的には下記のとおりです。

- **ネームスペース名** すべて小文字でアンダースコアを区切り文字としたスネークケース。
  - （例）word_and_word
- **クラス名／関数名／メソッド名** 小文字から始まるローワーキャメルケース。
  - （例）wordAndWord
- **例外** 標準ライブラリのクラスやメソッドと同質・同類・類似の場合、それがスネークケースであっても、同じ名前か同様の名前を用いる。
  - （例）push_backメソッド, lock_guardクラス, max_size_realメソッド（類似メソッド）
- **混在の許容** 一つのソースファイルやクラス内に、キャメルケースと例外（スネークケース）が混在することを問題とせず、むしろ独自拡張を識別可能なものとして扱う。
  - （例）dynamic_array::conrtainerクラスは、std::vectorに準拠し、assign, popo_back などのメソッドを実装しており、独自メソッドは assignArray のようなキャメルケースの名前で実装。

ライブラリのソースファイル

ソースファイルは、Visual C++とGCCの互換性のために、下記の仕様で統一しています。

- 文字コード ... UTF-8（BOM付き）
  - Visual Studio が扱う標準的なユニコードは UTF-8（BOM付き）です。
  - GCCが扱う標準的なユニコードは UTF-8（BOMなし）ですが、4.3.x 以降、BOM付きにも対応しています。
- 改行コード ... LF
  - Windows系OSの標準的な改行コードは CR+LF ですが、Visual StudioはLFのみのソースファイルも扱えます。
  - Unix系OSの標準的な改行コードは LF です。

ディレクトリ構成

サブモジュール／ライブラリ本体

```
[gasha_examples/gasha_proj]  ... ライブラリサンプルプログラム／ビルドプロジェクト用リポジトリ
 |
 |-[sub]                     ... サブモジュール用
    |
    `-[gasha]                ... サブモジュール：ライブラリ本体用リポジトリ
       |
```

```
      |-[include]           ... インクルードファイル用
      |  |
      |  `-[gasha]          ... ネームスペースディレクトリ
      |     |                   ※ライブラリファイル名の競合を避けるため
      |     |- *.h/*.inl                ... 各種インクルードファイル
      |     |
      |     `-[build_settings] ... ビルド設定用
      |        |
      |        `- build_settings.h      ... ライブラリビルド設定ファイル
      |
      |-[lib]               ... ライブラリファイル用
      |  |
      |  |-[vc]             ... Visual C++用
      |  |  |
      |  |  |- gasha_x86.lib            ... x86リリースビルド用
      |  |  |- gasha_x86_debug.lib      ... x86デバッグビルド用
      |  |  |- gasha_x64.lib            ... x64リリースビルド用
      |  |  `- gasha_x64_debug.lib      ... x64デバッグビルド用
      |  |
      |  `-[gcc]            ... GCC用
      |     |
      |     |- gasha_x86.a              ... x86リリースビルド用
      |     `- gasha_x86_debug.a        ... x86デバッグビルド用
      |                                     ※x86版Cygwinで開発したため、x64版は現状なし
      |
      `-[proj]
         |
         `- Makefile_common ... GCC用メイクファイルの共通部分
                                （ライブラリ／サンプルプログラム共用）
```

サブモジュール／ライブラリ挙動設定

```
[gasha_examples/gasha_proj]  ... ライブラリサンプルプログラム／ビルドプロジェクト用リポジトリ
 |
 |-[sub]                     ... サブモジュール用
    |
    `-[gasha_settings]       ... サブモジュール：ライブラリ挙動設定用リポジトリ
       |
       `-[include]           ... ライブラリインクルードファイル用
          |
          `-[gasha]          ... ネームスペースディレクトリ
             |                   ※ライブラリファイル名の競合を避けるため
             `-[build_settings] ... ビルド設定用
                |
                `-[project]     ... プロジェクト固有設定用
                   |
                   |- first_settings.h      ... プロジェクト固有のライブラリ設定（先行設定）
                   |                            ※ネームスペース設定など
                   `- last_setteings.h      ... プロジェクト固有のライブラリ設定
                                                ※SSE有効化設定など
```

サブモジュール／ライブラリソース

```
[gasha_examples/gasha_proj]  ... ライブラリサンプルプログラム／ビルドプロジェクト用リポジトリ
 |
 |-[sub]                     ... サブモジュール用
    |
    `-[gasha_src]            ... サブモジュール：ライブラリソース用リポジトリ
       |
       |-[proj]              ... ライブラリビルドプロジェクト用
       |  |
       |  |- standard.h/.cpp               ... 強制インクルード／プリコンパイル済みヘッダー
       |  |
       |  |- gasha.vcxproj                 ... Visual C++用ライブラリビルドプロジェクト
       |  |- Makefile                      ... GCC用ライブラリビルドメイクファイル
       |  `- mk.sh                         ... GCC用ライブラリビルドシェルスクリプト
       |
       `-[src]               ... ライブラリソースファイル用
          |-...
```

実行ファイル

```
[gasha_examples]             ... ライブラリサンプルプログラム用リポジトリ
 |
 |-[exe]                     ... 実行ファイル用
    |
    |- run_all.bat                        ... Visual C++ 版全サンプルプログラム一括実行＆実行ログ記録用バッチファイル
```

```
    |- run_all.sh                          ... GCC版全サンプルプログラム一括実行＆実行ログ記録用バッチファイル
    |
    |-[vc]                    ... Visual C++でビルドした実行ファイルの置き場
    |  |
    |  |- run_all.bat                       ... x86＆x64全サンプルプログラム一括実行＆実行ログ記録用バッチファイル
    |  |
    |  |-[x86]                ... x86(32bit)向け
    |  |  |
    |  |  |-（サンプル名）.exe              ... サンプルプログラム実行ファイル
    |  |  |- run_all.bat                     ... リリースビルド＆デバッグビルド全サンプルプログラム一括実行＆実行ログ記録用バッチファイル
    |  |  |
    |  |  |-[bat]             ... サンプルプログラム実行と実行ログ記録用
    |  |  |  |
    |  |  |  |-（サンプル名）.bat           ... サンプルプログラム個別実行＆実行ログ記録用バッチファイル
    |  |  |  |- run_all.bat                  ... 全サンプルプログラム一括実行＆実行ログ記録用バッチファイル
    |  |  |  |
    |  |  |  `-[log]
    |  |  |     |
    |  |  |     `-（サンプル名）.log        ... サンプルプログラム個別実行ログファイル
    |  |  |
    |  |  `-[Debug]           ... デバッグビルド用
    |  |     |-...                 ※親フォルダ（リリースビルド用）と同様の構成
    |  |
    |  `-[x64]                ... x64(64bit)向け
    |     |- ...                  ※x86版と同様の構成
    |
    `-[gcc]                   ... GCCでビルドした実行ファイルの置き場
       |
       `-[x86]                ... x86(32bit)向け
          |- ...                  ※Visual C++用と同様の構成
                                  ※シェルスクリプトの代わりにバッチファイル使用
                                  ※x86版Cygwinで開発したため、x64版は現状なし
```

各サンプルプログラムのソース

```
[gasha_examples]             ... ライブラリサンプルプログラム用リポジトリ
 |
 |-[src]                     ... サンプルプログラムソースファイル用
    |
    |-[（サンプル名）]       ... 各サンプルプログラムソースファイル用
    |  |
    |  |- *.cpp/.h
```

ライブラリおよび各サンプルプログラムのビルドプロジェクト

```
[gasha_examples]             ... ライブラリサンプルプログラム用リポジトリ
 |
 |-[proj]                    ... ライブラリ／サンプルプログラムビルドプロジェクト用
    |
    |-[（サンプル名）]       ... 各サンプルプログラムビルドプロジェクト用
    |  |
    |  |- main.cpp                         ... サンプルプログラム用メイン関数
    |  |- standard.h/.cpp                  ... サンプルプログラム用強制インクルード／プリコンパイル済みヘッダー
    |  |
    |  |- （サンプル名）.vcxproj           ... Visual C++用サンプルプログラムビルドプロジェクト
    |  |- Makefile                         ... GCC用サンプルプログラムビルドメイクファイル
    |  `- mk.sh                            ... GCC用サンプルプログラムビルドシェルスクリプト
    |
    |- Makefile_example_common             ... GCC用サンプルプログラムメイクファイルの共通部分
    |
    |- gasha_examples.sln                  ... Visual C++用ライブラリ＆全サンプルプログラムビルドソリューション
    `- mk_all.sh                           ... GCC用ライブラリ＆全サンプルプログラムビルドシェルスクリプト
```

ライブラリおよび各サンプルプログラムのビルドプロジェクト

```
[gasha_proj]                 ... ライブラリビルドプロジェクト用リポジトリ
 |
 |-[proj]                    ... ライブラリビルドプロジェクト用
    |
    |- gasha.sln                           ... Visual C++用ライブラリビルドソリューション
    `- mk.sh                               ... GCC用ライブラリビルドシェルスクリプト
```

ライブラリの利用手順

- **前提１：GASHAもしくはその派生ライブラリをgitリポジトリで管理するものとします。**
- **前提２：ライブラリを複数プロジェクトで共通利用するものとします。**

1. **ライブラリとライブラリ挙動設定リポジトリをプロジェクトに配置**

   サブモジュールとして、ライブラリ用のリポジトリを配置して下さい。

   `project/sub/gasha` ... ライブラリ本体用リポジトリ(https://github.com/gakimaru/gasha)
   `project/sub/gasha_settings` ... ライブラリ挙動設定用リポジトリ(https://github.com/gakimaru/gasha_settings)

   ライブラリ自体をプロジェクト向けにビルドする場合は、ライブラリソース用リポジトリも配置して下さい。

   `project/sub/gasha_src` ... ライブラリソース用リポジトリ(https://github.com/gakimaru/gasha_src)

2. **インクルードパスを設定**

   下記のパスをプロジェクトのインクルードパスに追加して下さい。

   `project/sub/gasha/include` ... ライブラリ用
   `project/sub/gasha_setting/include` ... ライブラリ挙動設定用

3. **ライブラリパスとライブラリファイルを設定**

   下記のいずれか一つのファイルをプロジェクトのライブラリファイルに追加して下さい。

   【Visual C++用】

   `project/sub/gasha/lib/vc/gasha_x86.lib` ... x86リリースビルド用
   `project/sub/gasha/lib/vc/gasha_x86_debug.lib` ... x86デバッグビルド用
   `project/sub/gasha/lib/vc/gasha_x64.lib` ... x64リリースビルド用
   `project/sub/gasha/lib/vc/gasha_x64_debug.lib` ... x64デバッグビルド用

   【GCC用】

   `project/sub/gasha/lib/gcc/gasha_x86.a` ... x86リリースビルド用
   `project/sub/gasha/lib/gcc/gasha_x86_debug.a` ... x86デバッグビルド用
   ※x86版Cygwinで開発したため、x64版は現状なし

4. **【推奨】強制インクルードとプリコンパイル済みヘッダーを設定**

   プラットフォーム／言語設定を暗黙的に全ソースファイルに反映させるために、「強制インクルード」を使用することを推奨します。
   更に、強制インクルードファイルを「プリコンパイル済みヘッダー」にすることで、コンパイル速度を高速化することを、合わせて推奨します。

   【設定例：standard.h】

   プロジェクトファイル（*.vcxproj, Makefile）と同じディレクトリに standard.h を配置して下さい。
   `stadard.h`の内容（この一行のみ）

   ```
   #include <gasha/build_settings/build_settings.h>
   ```

   【設定例：standard.cpp】 ※Visual C++のみ必要

   プロジェクトファイル（*.vcxproj, Makefile）と同じディレクトリに standard.cpp を配置して下さい。
   `stadard.cpp`の内容（この一行のみ）

   ```
   #include <standard.h>
   ```

   【Visual C++の場合】

   プロジェクトのプロパティから、［C/C++］→［詳細設定］ページの設定を下記のように変更して下さい。

   - ［必ず使用されるインクルードファイル］に standard.h を指定

   プロジェクトのプロパティから、［C/C++］→［プリコンパイル済みヘッダー］ページの設定を下記のように変更して下さい。

   - ［プリコンパイル済みヘッダー］に「使用(/Yu)」を指定
   - ［プリコンパイル済みヘッダーファイル］に standard.h を指定

> 更に、standard.cpp のプロパティから、［C/C++］→［プリコンパイル済みヘッダー］ページの設定を下記のように変更して下さい。
>
> > ・［プリコンパイル済みヘッダー］に「作成(/Yc)」を指定
> > ・［プリコンパイル済みヘッダーファイル］に standard.h を指定
>
> **【GCCの場合】**
>
> > g++コマンドでプリコンパイル済みヘッダーファイル standard.h.gch を作成して下さい。
> > 【例】
> >
> > > $ g++ （-std=c++11 や -g などのコンパイルオプション） -x c++-header standard.h
> >
> > standard.h.gch はインクルードパスが通った場所に配置して下さい。
> >
> > コンパイル時には、g++コマンドに -inlude オプションを指定して下さい。
> > .h.gch ではなく、.h ファイルを指定します。
> > 【例】
> >
> > > $ g++ （-std=c++11 や -g などのコンパイルオプション） -include standard.h -c xxx.cpp -o xxx.o

5. **サブモジュールのブランチ（もしくはタグ／バージョン）を確定してコミット**

> サブモジュールの挙動設定ファイルの変更やライブラリのビルドなどを行った後、プロジェクトをコミットして下さい。
>
> サブモジュール用のフォルダがコミット対象になります。
>
> これにより、プロジェクトで使用するサブモジュール（ライブラリのリポジトリ）のバージョンが設定されます。

サンプルプログラム用プロジェクト

サンプルプログラム用プロジェクト

```
[gasha_examples]  ... ライブラリサンプルプログラム用リポジトリ
 |
 `-[proj]
    |
    |-[example_build_settings]      ... ビルド設定テスト
    |-[example_utility]             ... 汎用ユーティリティテスト
    |-[example_type_traits]         ... 型特性ユーティリティテスト
    |-[example_basic_math]          ... 基本演算処理テスト
    |-[example_crc32]               ... CRC32計算処理テスト
    |-[example_sort_and_search]     ... ソート＆探索処理テスト
    |-[example_shared_data]         ... マルチスレッド共有データ／ロックフリーアルゴリズムテスト
```

> 各サンプルプログラムは、チュートリアル的な内容ではなく、開発過程で機能やパフォーマンスの評価に使用したものそのままです。
> 条件を変えながら多彩なテストをできるように構成しているものも多く、難解に見えるかもしれません。
> ライブラリを習得いただく際は、関連資料とソースコード（主にヘッダーファイル）のコメントを合わせてご確認頂ければ幸いです。

免責事項

本ライブラリの実績は不十分であり、品質を保証できる状態ではありません。
ライブラリの不具合には基本的に対応できませんので、ご利用の際は、自己責任でお願いします。
なお、MITライセンスに基づき、自由な改変や商用利用も可能ですので、独自に改変・修正の上でご利用頂ければ幸いです。

関連資料

ゲームシステムのアーキテクチャと開発環境（GitHub）
ゲームシステムのアーキテクチャと開発環境（DropBox）

> 資料にはサンプルプログラムも多数掲載しており、ライブラリのプログラムと同じものも含んでいます。
> サンプルプログラム作成とそれに基づく資料作成が先行し、その後ライブラリにまとめ直しているので、両者のソースコードには多少の相違があります。

■■以上