

Generator of memory manager and Database-Oriented framework

What is GommaDof?

GommaDofは拡張BNFから安全で効率のよい様々なデータ構造とそれらを扱うための関数を自動生成するコードジェネレータです。GommaDofを導入することで、ソフトウェア技術者はバグやパフォーマンス低下を生じやすいメモリまわりの煩雑な管理から解放され、開発の本質となる処理の設計・実装に注力できるようになります。

GommaDofはメモリ使用効率が高く、検索速度に優れたデータ構造を生成し、カスタマイズによりコンテナ型データの構造や検索キーを最適化することができます。また、メモリ確保・解放、文字列やTLV(type length value)構造との相互変換など、メモリに対する操作としてよく用いられる必要性の高い関数を生成することができます。拡張BNFの記述を変更するだけで、データ構造と関数を一括して自動生成できるため、開発時の問題の切り分けも容易になり、バグの発生を低減します。また、仕様への追従が早く、Rapid Prototypingに有用で、仕様変更にも柔軟な対応が可能になります。

GommaDofのメリット

- ・文字列とデータの相互変換による開発・デバッグコストの低減。
- ・効率のよいデータ構造。
- ・バグ発生を低減。
- ・メモリ管理が分離されて問題の切り分けが容易に。
- ・操作関数の記述が不要。
- ・拡張BNFを記述するだけで簡単。

コスト低減・開発期間短縮

可読性の高い 安全で効率のよいプログラムコード

- ・基本データ構造
- ・文字列、データ間変換
- ・ネットワークパケット変換
- ・カプセル化コード
- ・メモリ管理(GC)コード
- ・コンテナ構造
- ・高速検索データ構造
- ・検索関数
- ・挿入・削除関数

BNF記述



GommaDofはC言語のソースコードを生成します。他の言語についてはご相談ください。

Why do we need GommaDof?

開発現場では次のような問題がしばしば発生し、開発の効率が下がり、開発期間・コストの増大に繋がっています。

- ・何度も同じようなメモリ管理のコードを書いている
- ・データ構造を変更したら関数も全部作り直し。
- ・メモリ確保・解放の整合性がとれない。
- ・無駄なメモリ操作で性能が上がらない。
- ・毎度同じようなコードを書くことに疑問を感じる。

このような問題は **GommaDof** が解決します！

Where does GommaDof work?

- ・ネットワーク機器、ルーティングプロトコルの設計・開発
- ・各種組み込み機器、制御関連部分設計・開発
- ・エンドユーザ向けソフトウェア、仕様変更が多く発生するソフトウェアの設計・開発
- ・短期間での試作・評価

How does GommaDof work?

```
Address ::= "IPv4" uint8 uint8 uint8 uint8
          | "IPv6" uint16 uint16 uint16 uint16 uint16 uint16 uint16 uint16;
Nodes ::= Address*;
Segment ::= "Seg" Address Nodes; -- "Seg" (GW addr) (other nodes)
```

わずか4行のBNFを入力



3800行超のデータ構造・関数を生成

生成例は裏面 >>

コードの生成例

おもて面の BNF から以下のようなソースコードを生成します。

3884 行 127,573Byte
これらを手で書くと大変です！

生成されるファイル

00EnumTag.sh	Address.h	GCfunc.c	Makefile	Segment.h
00Primit.c	GCapi.h	GCfunc.h	Nodes.c	
00Primit.h	CEngine.c	GTstring.c	Nodes.h	
Address.c	CEngine.h	GTstring.h	Segment.c	

ソースコードの一部

Address.h

```
#ifndef Address_H_INCLUDED
#define Address_H_INCLUDED
struct GC_context;
union Address;

typedef void (*IPv4_apply_func_t)(struct GC_context *, unsigned char, unsigned char, unsigned char, unsigned char);
typedef void (*IPv6_apply_func_t)(struct GC_context *, unsigned short, unsigned short, unsigned short, unsigned short, unsigned short, unsigned short, unsigned short, unsigned short);

#define Address_apply_(cx, obj, prefix) \
    (Address_apply(cx, obj) \
     , prefix##IPv4 \
     , prefix##IPv6 \
    )

void Address_apply(struct GC_context *cx, union Address *x, IPv4_apply_func_t func_IPv4, IPv6_apply_func_t func_IPv6);
void Address_delete(struct GC_context *cx, union Address *x);
unsigned char Address_get_IPv4_m1(struct GC_context *cx, union Address *x);
unsigned char Address_get_IPv4_m2(struct GC_context *cx, union Address *x);
unsigned char Address_get_IPv4_m3(struct GC_context *cx, union Address *x);
unsigned char Address_get_IPv4_m4(struct GC_context *cx, union Address *x);
unsigned short Address_get_IPv6_m1(struct GC_context *cx, union Address *x);
unsigned short Address_get_IPv6_m2(struct GC_context *cx, union Address *x);
unsigned short Address_get_IPv6_m3(struct GC_context *cx, union Address *x);
unsigned short Address_get_IPv6_m4(struct GC_context *cx, union Address *x);
unsigned short Address_get_IPv6_m5(struct GC_context *cx, union Address *x);
unsigned short Address_get_IPv6_m6(struct GC_context *cx, union Address *x);
unsigned short Address_get_IPv6_m7(struct GC_context *cx, union Address *x);
unsigned short Address_get_IPv6_m8(struct GC_context *cx, union Address *x);
union Address * Address_getroot_IPv4(struct GC_context *cx);
union Address * Address_getroot_IPv6(struct GC_context *cx);
enum Tags Address_gettag(struct GC_context *cx, union Address *x);
union Address * Address_new_IPv4(struct GC_context *cx);
union Address * Address_new_IPv6(struct GC_context *cx);
union Address * Address_newset_IPv4(struct GC_context *cx, unsigned char m1, unsigned char m2, unsigned char m3, unsigned char m4);
union Address * Address_newset_IPv6(struct GC_context *cx, unsigned short m1, unsigned short m2, unsigned short m3, unsigned short m4, unsigned short m5, unsigned short m6, unsigned short m7, unsigned short m8);
const char * Address_read(struct GC_context *cx, const char *str, union Address **x, void *cont0);
void Address_set_IPv4_m1(struct GC_context *cx, union Address *x, unsigned char arg);
void Address_set_IPv4_m2(struct GC_context *cx, union Address *x, unsigned char arg);
void Address_set_IPv4_m3(struct GC_context *cx, union Address *x, unsigned char arg);
void Address_set_IPv4_m4(struct GC_context *cx, union Address *x, unsigned char arg);
void Address_set_IPv6_m1(struct GC_context *cx, union Address *x, unsigned short arg);
void Address_set_IPv6_m2(struct GC_context *cx, union Address *x, unsigned short arg);
void Address_set_IPv6_m3(struct GC_context *cx, union Address *x, unsigned short arg);
void Address_set_IPv6_m4(struct GC_context *cx, union Address *x, unsigned short arg);
void Address_set_IPv6_m5(struct GC_context *cx, union Address *x, unsigned short arg);
void Address_set_IPv6_m6(struct GC_context *cx, union Address *x, unsigned short arg);
void Address_set_IPv6_m7(struct GC_context *cx, union Address *x, unsigned short arg);
void Address_set_IPv6_m8(struct GC_context *cx, union Address *x, unsigned short arg);
struct GTstring * Address_show(struct GC_context *cx, union Address *x);
unsigned int Address_tvdecode(struct GC_context *cx, const char *str, union Address **x);
struct GTstring * Address_tvencode(struct GC_context *cx, union Address *x, unsigned int *length);
#endif /* !Address_H_INCLUDED */
```

Nodes.h

```
#ifndef Nodes_H_INCLUDED
#define Nodes_H_INCLUDED
struct GC_context;
union Address;
struct Nodes;

typedef void (*Nodes_foreach_func_t)(struct GC_context *, union Address *);
void Nodes_add(struct GC_context *cx, struct Nodes *base, union Address *x);
void Nodes_addlist(struct GC_context *cx, struct Nodes *left, struct Nodes *right);
void Nodes_delete(struct GC_context *cx, struct Nodes *x);
void Nodes_foreach(struct GC_context *cx, struct Nodes *x, Nodes_foreach_func_t func);
union Address * Nodes_get(struct GC_context *cx, struct Nodes *x, int n);
struct Nodes * Nodes_getroot(struct GC_context *cx);
int Nodes_length(struct GC_context *cx, struct Nodes *x);
struct Nodes * Nodes_new(struct GC_context *cx);
const char * Nodes_read(struct GC_context *cx, const char *str, struct Nodes **x, void *cont0);
void Nodes_remove(struct GC_context *cx, struct Nodes *base, union Address *x);
void Nodes_removeall(struct GC_context *cx, struct Nodes *base);
void Nodes_set(struct GC_context *cx, struct Nodes *x, int n, union Address *arg);
struct GTstring * Nodes_show(struct GC_context *cx, struct Nodes *x);
unsigned int Nodes_tvdecode(struct GC_context *cx, const char *str, struct Nodes **x);
struct GTstring * Nodes_tvencode(struct GC_context *cx, struct Nodes *x, unsigned int *length);
#endif /* !Nodes_H_INCLUDED */
```

Segment.h

```
#ifndef Segment_H_INCLUDED
#define Segment_H_INCLUDED
struct GC_context;
union Address;
union Segment;

typedef void (*Seg_apply_func_t)(struct GC_context *, union Address *, struct Nodes *);
#define Segment_apply_(cx, obj, prefix) \
    (Segment_apply(cx, obj) \
     , prefix##Seg \
    )

void Segment_apply(struct GC_context *cx, union Segment *x, Seg_apply_func_t func_Seg);
void Segment_delete(struct GC_context *cx, union Segment *x);
union Address * Segment_get_Seg_m1(struct GC_context *cx, union Segment *x);
struct Nodes * Segment_get_Seg_m2(struct GC_context *cx, union Segment *x);
union Address * Segment_get_m1(struct GC_context *cx, union Segment *x);
struct Nodes * Segment_get_m2(struct GC_context *cx, union Segment *x);
union Segment * Segment_getroot_Seg(struct GC_context *cx);
enum Tags Segment_gettag(struct GC_context *cx, union Segment *x);
union Segment * Segment_new_Seg(struct GC_context *cx);
union Segment * Segment_newset_Seg(struct GC_context *cx, union Address *m1, struct Nodes *m2);
const char * Segment_read(struct GC_context *cx, const char *str, union Segment **x, void *cont0);
void Segment_set_Seg_m1(struct GC_context *cx, union Segment *x, union Address *arg);
void Segment_set_Seg_m2(struct GC_context *cx, union Segment *x, struct Nodes *arg);
void Segment_set_m1(struct GC_context *cx, union Segment *x, union Address *arg);
void Segment_set_m2(struct GC_context *cx, union Segment *x, struct Nodes *arg);
struct GTstring * Segment_show(struct GC_context *cx, union Segment *x);
unsigned int Segment_tvdecode(struct GC_context *cx, const char *str, union Segment **x);
struct GTstring * Segment_tvencode(struct GC_context *cx, union Segment *x, unsigned int *length);
#endif /* !Segment_H_INCLUDED */
```

Address.c (抜粋)

```
#include <assert.h>
#include <ctype.h>
#include <string.h>
#include "00EnumTags.h"
#include "Address.h"
#include "Primit.h"
struct IPv4 {
    unsigned char m1;
    unsigned char m2;
    unsigned char m3;
    unsigned char m4;
};
struct IPv6 {
    unsigned short m1;
    unsigned short m2;
    unsigned short m3;
    unsigned short m4;
    unsigned short m5;
    unsigned short m6;
    unsigned short m7;
    unsigned short m8;
};
union Address {
    struct IPv4 m1;
    struct IPv6 m2;
};
```

Nodes.c (抜粋)

```
#include <assert.h>
#include <ctype.h>
#include "00EnumTags.h"
#include "Address.h"
#include "Nodes.h"
struct Nodes {
    struct Nodes *link;
    union Address *m1;
};
```

Segment.c (抜粋)

```
#include <assert.h>
#include <ctype.h>
#include <string.h>
#include "00EnumTags.h"
#include "Address.h"
#include "Nodes.h"
#include "Segment.h"
struct Seg {
    union Address *m1;
    struct Nodes *m2;
};
union Segment {
    struct Seg m1;
};
```

株式会社トランス・ニュー・テクノロジー (TNT) は「アーキテクチャへの挑戦」を標榜し、研究型開発を主業務にしています。分野もコンパイラやネットワークプロトコルの研究開発から並列数値計算等、アプリケーションレベルにとどまらない広い範囲の技術を提供しています。

常に本当の意味での「時代の最先端」、さらにはその一歩先を目指し、新技術を開拓していく、それが TNT です。



株式会社トランス・ニュー・テクノロジー

〒116-0013 東京都荒川区西日暮里 5-14-4 KYビル 8F

Tel: 03-5604-1188 Fax: 03-5604-1199

<http://www.trans-nt.com/>

お問い合わせ先

info@trans-nt.com

担当：木村・中野