

目次

1 データの検証	1
1.1 式	2
1.2 あいまいな式	2
1.3 プログラム変数	2
1.4 人工配列	2
1.5 出力フォーマット	3
1.6 メモリ検査	3
1.7 メモリタグ	3
1.8 自動表示	4
1.9 表示設定	4
1.10 きれいな表示	6
1.10.1 pretty-printer 導入	6
1.10.2 pretty-printer の例	6
1.10.3 pretty-printer のコマンド	7
1.11 値の履歴	7
1.12 コンビニエンス変数	7
1.13 コンビニエンス関数	9
1.14 レジスタ	10
1.15 浮動小数点ハードウェア	10
1.16 ベクタユニット	10
1.17 OS 情報	10
1.18 メモリ領域属性	11
1.18.1 属性	11
1.18.1.1 メモリアクセスモード	11
1.18.1.2 メモリアクセスサイズ	11
1.18.1.3 データキャッシュ	11
1.18.1.4 メモリアクセスチェック	11
1.19 メモリーファイル間のコピー	12

1 データの検証

```
print [[<options>] --] <expr>
print [[<options>] --] /<f> <expr>
```

式 `expr` の値を表示します。指定しない場合は、最後の値を表示します。 `options` に指定できる値は以下です。

- `--address [on|off]`: アドレスを表示します。
- `-array [on|off]`: 配列のフォーマットをします。
- `-array-indexes [on|off]`: 配列の添え字を表示します。
- `-characters <number-of-characters>|elements|unlimited`: 表示する文字列の長さを制限します。
- `-elements <number-of-elements>|unlimited`: 表示する配列の要素数を指定します。

- `-max-depth <depth>|unlimited`: ネストした構造を表略する閾値を設定します。
- `-nibbles [on|off]`: バイナリを 4 ビットグループで表示するかどうか
- `-memory-tag-violations [on|off]`: メモリタグ違反に関する情報を表示する。
- `-null-stop [on|off]`: 文字配列の表示を NULL 文字で停止するかどうか。
- `-object [on|off]`: C++ の virtual function テーブルの表示設定
- `-pretty [on|off]`: 構造体の pretty フォーマットの設定
- `-repeats <number-of-repeats>|unlimited`: 繰り返しの要素表示の閾値の設定
- `-static-member [on|off]`: C++ の static メンバの表示設定
- `-symbol [on|off]`: ポインタ表示時のシンボル名の表示設定
- `-union [on|off]`: 構造体内部の共用体の表示設定
- `-vtbl [on|off]`: C++ の virtual function テーブルの表示設定

1.1 式

`print` などのコマンドで引数にとる式は使用中の言語の任意の式を受け付けます。

その他にも言語に依存せず以下の演算が用意されています。

@

メモリの一部を配列として扱います。

::

変数が定義されているファイルや関数を指定して変数を指定します。

{<type>} <addr>

`addr` の値のメモリ位置を `type` 型として解釈します。

1.2 あいまいな式

`set multiple-symbols <mode>`

式があいまいな場合の動作を設定します。設定できるものは以下の通りです。

- `all`: デフォルト。全ての選択肢を選択します。一位に選ぶ必要がある場合、メニューが表示されます。
- `ask`: あいまいさがある場合、常にメニューを表示します。
- `cancel`: あいまいさがある場合、エラーをはいてコマンドが中断されます。

1.3 プログラム変数

式の変数は選択中のスタックフレームで解釈されます。

<file>::<var>

とすると他の場所の変数も指定できます。

1.4 人工配列

<arr>@<len>

の形で `arr` を最初の要素とする長さ `len` の配列として `&arr` からのメモリを表示します。

キャストでも同様の動作をさせることはできます。

1.5 出力フォーマット

デフォルトでは GDB は型に沿って値を成型して表示します。一方、フォーマットを指定して表示することもできます。

x

バイナリを 16 進数で表示します。

d

バイナリを 10 進数で表示します

u

バイナリを符号なし 10 進数で表示します。

o

バイナリを 8 進数で表示します。

t

バイナリを 2 進数で表示します。

a

アドレスを表示します。

c

値を整数値にキャストして文字列として表示します。

f

浮動小数として表示します。

s

可能であれば文字列として扱います。

z

ゼロ埋めされた 16 進数で表示します。

r

raw フォーマットで表示します。

1.6 メモリ検査

x[/[<n>]][<f>][<u>]] [<addr>]

- n: 表示するメモリ量(単位は u で指定)を指定
- f: フォーマットを指定
- u: 単位を指定。指定できるもの:
 - b: Byte
 - h: Halfwords(2Bytes)
 - w: words(4Bytes)
 - g: Giant words(8Bytes)
- addr: 開始アドレス

1.7 メモリタグ

メモリ・タグは、ポインタを介したメモリ・アクセスを検証するために 1 対のタグを使用するメモリ保護技術である。タグは、アーキテクチャにもよるが、通常は数ビットからなる整数値である。

1.8 自動表示

ある式の値を頻繁に表示したい場合、自動表示が利用できます。

```
display[/<fmt>] <expr>|<addr>
```

自動表示リストに `expr` を追加します。

```
undisplay <dnums>...
```

自動表示リストから削除します。

```
disable display <dnums>...
```

自動表示を無効化します。

```
enable display
```

自動表示を有効化します。

```
display
```

現在のリストの上の式の値を表示します。

```
info display
```

自動表示リストを表示します。値は表示しません。

1.9 表示設定

GDB は表示方法について以下の設定を提供しています。

```
set print address [on|off]
```

スタックトレース、構造体の値、ポインタの値、ブレークポイントなどの場所を示すメモリアドレスを表示します。デフォルト値は `on` です。

```
set print symbol-filename [on|off]
```

`on` のときシンボルのソースファイル名と行番号をアドレスのシンボル形式で表示します。

```
set print max-symbolic-offset <max-offset>|unlimited
```

シンボリックアドレスを表示するオフセットの最大値を設定します。最大値以上のオフセットの場合は表示されません。0 と `unlimited` は等価で、前にシンボルがある限り常に表示します。

```
set print symbol [on|off]
```

あるアドレスに対応するシンボルがあれば、それを表示します。

```
set print array [on|off]
```

配列をきれいに整形して表示します。デフォルト値は `off` です。

```
set print array-indexed [on|off]
```

配列を表示する際に書く要素のインデックスを憑依します。デフォルト値は `off` です。

```
set print nibbles [on|off]
```

`print` コマンドを `/t` で表示する場合に 4bit で区切って表示します。

```
set print characters <number-of-characters>|elements|unlimited
```

GDB が表示する文字列の制限を設定します。`elements` を設定すると配列の大きさを分表示します。デフォルト値は `elements` です。

```
set print elements <number-of-elements>|unlimited
```

GDB が表示する配列の要素数の上限を設定します。デフォルト値は 200 です。

```
set print frame-arguments <value>
```

フレームを表示するときに引数の値をどのように表示するかを設定します。`value` に設定できる値は以下のとおりです:

- `all`: すべての引数が表示されます
- `scalars`: スカラー値の引数のみ表示します
- `none`: どの引数の値も表示しません。値は...で置き換えられます
- `presence`: 引数がある場合は...が、ない場合は何も表示されません

デフォルト値は `scalars` です。

```
set print raw-frame-arguments [on|off]
```

フレームの引数をきれいに整形されていない生の状態で表示します。

```
set print entry-values <value>
```

関数エントリ時のフレーム引数の値の表示を設定します。value に指定できる値は以下のとおりです:

- `no`: 実際のパタメータ値のみ表示し、エントリポイントからの値は表示しません
- `only`: エントリポイントからの値のみ表示し、実際の値は表示しません
- `preferrrd`: エントリポイントからの値を表示し、それが不明で実際の値が既知の場合それを表示します
- `if-needed`: 実際の値を表示し、それが既知でない場合エントリポイントからの値が既知ならばそれを表示します
- `both`: 常にエントリポイントからの値と実際の値の両方を表示します
- `compact`: 実際の値およびエントリポイントからの値のうち既知の値を表示します。どちらも未知の場合 `optimized out` を表示します。MI モードでない場合、両方の値が既知であれば短縮表記 `param=param@entry=VALUE` を表示します
- `default`: 常に実際の値を表示します。エントリポイントからの値も既知の場合それらも表示します。MI モードでなければ短縮表記を使用します。

デフォルト値は `default` です。

```
set print frame-info <value>
```

フレームを表示するときに表示される情報を制御します。value に設定できる値は以下のとおりです:

- `short-location`: フレームレベル、PC(ソース行の先頭でなければ)、関数、引数を表示します
- `location`: `short-location` に加えソースファイルと行番号も表示します
- `location-and-address`: `location` に加え、ソース行の先頭でも PC を表示します
- `source-line`: PC(ソースの先頭でなければ)、行番号、ソース行を表示します
- `source-and-location`: `location` と `source-line` を表示します
- `auto`: 使用するコマンドによって自動的に表示される情報を決定します。

デフォルト値は `auto` です。

```
set print repeats <number-of-repeats>|unlimited
```

配列の繰り返し要素の表示を抑制する閾値を設定します。配列の連続した同一要素の数が閾値を超えると `<repeats n times>` を表示します。0 と `unlimited` は同等です。デフォルト値は 10 です。

```
set print max-depth <depth>|unlimited
```

ネストした構造体を省略記号に置き換える深さの閾値を設定します。

```
set print memory-tag-violations [on|off]
```

ポインタとアドレスを表示するときメモリタグ違反に関する情報を表示します。

```
set print null-stop [on|off]
```

文字列を表示するときに最初の NULL で表示を停止するかどうかの設定です。デフォルト値は off です。

```
set print pretty [on|off]
```

構造体を整形して表示するかどうかの設定です。

```
set print raw-values [on|off]
```

値のための整形を行わず生の値を表示するかどうかの設定です。

```
set print sevenbit-strings [on|off]
```

8bit 文字を \nnn という形式で表示します。

```
set print union [on|off]
```

構造体や他のユニオンに含まれるユニオンを表示するかどうかの設定です。デフォルトは on です。

以下は C++ 関連の設定です。

```
set print demangle [on|off]
```

C++ の名前を型安全リンケージのためにアセンブラやリンクに渡す mangle 形式ではなくソース形式で表示します。デフォルト値は on です。

```
set print asm-demangle [on|off]
```

アセンブラコードの表示時に mangled 形式でなくソース形式で表示します。デフォルト値は off です。

```
set demangle-style [<style>]
```

C++ の名前を表現するためにエンコーディングスキームを選択します。style を省略すると設定可能なスタイルのリストが表示されます。デフォルト値は auto です。

```
set print object [on|off]
```

オブジェクトへのポインタを表示する場合、仮想関数テーブルを使用して、宣言された型ではなく、オブジェクトの実際の型を識別します。

```
set print static-members [on|off]
```

C++ オブジェクトの静的メンバを表示します。デフォルト値は on です。

```
set print pascal_static-members [on|off]
```

パスカルのオブジェクトを表示するときに静的メンバを表示します。デフォルト値は on です。

```
set print vtbl [on|off]
```

C++ 仮想関数テーブルをきれいに表示します。

1.10 きれいな表示

1.10.1 pretty-printer 導入

GDB が値を表示するとき、まずその値に対応する pretty-printer が登録されているか確認します。あればそれ呼び出し、なければ通常通り表示されます。

1.10.2 pretty-printer の例

C++ の `std::string` は pretty-printer なしでは

```
$1 = {  
  static npos = 4294967295,  
  _M_dataplus = {
```

```

<std::allocator<char>> = {
  <__gnu_cxx::new_allocator<char>> = {
    <No data fields>}, <No data fields>
  },
  members of std::basic_string<char, std::char_traits<char>,
    std::allocator<char> >::_Alloc_hider:
  _M_p = 0x804a014 "abcd"
}
}

```

のように表示されます。これを pretty-printer に通すと
`$2 = "abcd"`
 と表示されます。

1.10.3 pretty-printer のコマンド

```

info pretty-printer [object-regex] [name-regex]

```

インストールされている pretty-printer を名前とともにリストアップします。

```

disable pretty-printer [object-regex] [name-regex]
enable pretty-printer [object-regex] [name-regex]

```

pretty-printer を無効化、有効化します。

1.11 値の履歴

print コマンドで出力された値は GDB の値履歴に保存されます。これにより他の式で値を参照することができます。値はシンボルテーブルが再読み込みまたは破棄されるまで保持されます。

過去の値は `$<num>` で参照できます。`$$<num>` とすれば最後の履歴から逆順に参照できます。単に `$` とすれば最新の履歴を参照します。

```
show values [<n>|+]
```

最後の十個の履歴を表示します。このコマンドは履歴を変更しません。数値を指定するとその番号から 10 個を、+ を指定すると前回表示した続きの 10 個を表示します。

1.12 コンビニエンス変数

GDB は、GDB の内部で値を保持し、参照するために使用できるコンビニエンス変数を提供します。

事前に定義されていない限り `$` から始まる名前なら何でもコンビニエンス変数になる。

```
set $foo = *obj_ptr
```

のように変数をセットできます。型はありません。

```
show convinience
```

コンビニエンス変数の一覧を表示します

```
init-if-undefined $<var> = <expression>
```

コンビニエンス変数がまだ定義されていなければ `expression` で初期化します。

以下は事前に定義されるコンビニエンス変数です。

`$_`

x コマンドで自動的に最後に調べたアドレスに設定されます。`$_`へのポインタです。

`$_`

`$_`の指すアドレスの値です。

`$_exitcode`

デバッグ対象のプログラムが正常終了すると終了コードに設定されます。このとき `$_exitsignal` を void にリセットします。

`$_exitsignal`

キャッチされなかったシグナルによってデバッグ対象のプログラムが終了すると、そのシグナルの番号が設定されます。このとき `$_exitcode` を void にリセットします。

`$_exception`

例外のキャッチポイントでスローされる例外オブジェクトが設定されます。

`$_ada_exception$`

Ada の例外キャッチポイントでスローされる例外のアドレスがセットされます。

`$_probe_argc`

`$_probe_arg0 ... $_probe_arg11`

静的プローブの引数です。

`$_sdata`

追加で収集された静的トレースポイントデータが格納されます。

`$_siginfo`

追加のシグナル情報が格納されます。

`$_tlb`

MS-Windows 上でネイティブモードで作動しているアプリケーションをデバッグする場合または `qGetTIBAddr` リクエストをサポートしている `GDBServer` に接続している場合スレッド情報ブロックのアドレスが格納されます。

`$_inferior`

現在の inferior の数が格納されます。

`$_thread`

現在のスレッドのスレッド番号が格納されます。

`$_gthread`

現在のスレッドのグローバルスレッド番号が格納されます。

`$_inferior_thread_count`

現在の inferior でアクティブなスレッドの数が格納されます。

`$_gdb_major`

`$_gdb_minor`

GDB のメジャー、マイナーバージョン番号が格納されます。

`$_shell_exitcode`

`$_shell_exitsignal`

シェルコマンドの exitcode および exitsignal が格納されます。

1.13 コンビニエンス関数

コンビニエンス引数に似たコンビニエンス関数もあります。これらも式中で使うことができますが、GDB の内部で実装されています。

`$_isvoid(<expr>)`

`expr` が `void` かどうか判定します。

`$_gdb_setting_str(<setting>)`

GDB の設定の内容を文字列として取得します。

`$_gdb_setting(<setting>)`

GDB の設定の内容を取得します。型は設定によります。

`$_gdb_maint_setting_str(<setting>)`

メンテナンス設定変数版の `$_gdb_setting_str` です。

`$_gdb_maint_setting(<setting>)`

メンテナンス設定変数版の `$_gdb_setting` です。

`$_shell(<command_string>)`

シェルコマンドを実行します。

`$_memeq(<buf1>, <buf2>, <length>)`

`buf1`, `buf2` で与えられるアドレスの長さ(Byte)が `length` に等しいかどうかを判定します。

`$_regex(<str>, <regex>)`

文字列 `str` が正規表現 `regex` にマッチするかどうかを判定します。

`$_streq(<str1>, <str2>)`

2つの文字列が等しいかどうかを判定します。

`$_strlen(<str>)`

文字列の長さを返します。

`$_caller_is(<name>[, <number_of_frames>])`

呼び出し元の関数名が `name` であるかどうかを判定します。`number_of_frames` を指定すると特定のスタックフレームに注目します。指定しない場合は1が入ります。

`$_caller_matches(<regex>[, <number_of_frames>])`

`$_caller_is` の正規表現版です。

`$any_caller_is(<name>[, <number_of_frames>])`

呼び出し元の関数に `name` があるかどうかを判定します。

`$any_caller_match(<name>[, <number_of_frames>])`

`$any_caller_is` の正規表現版です。

`$_as_string(<value>)`

非推奨の関数です。代わりに `%v` の仕様が推奨されます。`value` の文字列表現を返します。

`$_cimag(<value>)`

`$_creal(<value>)`

複素数値の実部、虚部を返します。

`help function`

コンビニエンス関数の一覧を表示します。

1.14 レジスタ

```
info registers [<reggroup>...|<regname>...]
```

浮動小数点レジスタとベクタレジスタを除くすべてのレジスタの名前と値を表示します。 `reggroup` を指定した場合、そのグループのレジスタに関して表示を行います。指定できるグループの一覧は `maint print reggroups` で確認できます。 `regname` を指定した場合、その名前のレジスタに関して表示を行います。

```
info all-registres
```

すべてのレジスタの名前と値を表示します。

1.15 浮動小数点ハードウェア

```
info float
```

浮動小数点ユニットに関するハードウェア依存の情報を表示します。

1.16 ベクタユニット

```
info vector
```

ベクタユニットに関する情報を表示します。

1.17 OS 情報

```
info auxv
```

実行中のプロセスまたはコアダンプファイルである `inferior` の補助ベクタを表示します。

```
info os [<infotype>]
```

OS の情報を表示します。 `infotype` には GNU/Linux では以下の値を指定できます。

- `cpus` : すべての CPU、コアのリストを表示します。
- `files` : ターゲット上でオープンされているファイルディスクリプタのリストを表示します。
- `modules` : ターゲットにロードされているすべてのカーネルモジュールのリストを表示します。
- `msg` : ターゲット上のすべての System V メッセージキューのリストを表示します。
- `processes` : ターゲット上のプロセスのリストを表示します。
- `procgroups` : ターゲット上のプロセスグループのリストを表示します。
- `semaphores` : ターゲット上のすべての System V セマフォセットのリストを表示します。
- `shm` : ターゲット上のすべての System V 共有メモリ領域のリストを表示します。
- `sockets` : ターゲット上のすべてのインターネットドメインソケットのリストを表示します。
- `threads` : ターゲット上で実行中のスレッドのリストを表示します。

`infotype` を指定しない場合、すべての OS 情報が表示されます。

1.18 メモリ領域属性

`mem <lower> <upper> <attributes>...`

`lower` と `upper` で囲まれたメモリ領域を `attributes...` で定義し、GDB が監視する領域のリストに追加します。

`mem auto`

メモリ領域に対するユーザの変更をすべて破棄し、ターゲットが提供する領域があればそれを利用し、サポートしていない場合は使用しません。

`delete mem <nums>...`

GDB が監視する領域リストからメモリ領域を削除します。

`disable mem <nums>...`

`enable mem <nums>...`

GDB が監視する領域リストのメモリ領域の監視を無効化/有効化します。

`info mem`

定義されたすべてのメモリ領域の一覧を表示します。

1.18.1 属性

1.18.1.1 メモリアクセスモード

メモリアクセスモードは GDB がメモリ領域に対して行うアクセスのモードです。

`ro`

`w0`

`rw`

読み込み専用/書き込み専用/読み書きのモードです。デフォルト値は `rw` です。

1.18.1.2 メモリアクセスサイズ

`8`

`16`

`32`

`64`

指定した bit のメモリアクセスサイズを持ちます。

1.18.1.3 データキャッシュ

GDB がターゲットメモリをキャッシュするかどうかの設定です。

`cache`

`nocache`

キャッシュあり/なしです。

1.18.1.4 メモリアクセスチェック

`set mem inaccessible-by-default [on|off]`

メモリ範囲に明示的に記述されていないメモリ領域へのアクセスを拒否するかどうかの設定です。デフォルト値は `on` です。

1.19 メモリーファイル間のコピー

```
dump [format] memory <filename> <start_addr> <end_addr>
```

```
dump [format] value <filename> <expr>
```

メモリの start_addr から end_addr までの内容または expr の値を与えられたフォーマットで filename にダンプします。指定できるフォーマットは以下のとおりです。

- binary : 生のバイナリです。
- ihex : Intel Hex format です。
- srec : Motorola S-record format です。
- tekhex : Tektronix Hex format です。
- verilog : Verilog Hex format です。

```
append [format] memory <filename> <start_addr> <end_addr>
```

```
append [format] value <filename> <expr>
```

メモリの start_addr から end_addr までの内容または expr の値を与えられたフォーマットで filename に追加します。

```
restore <filename> [binary] <bias> <start> <end>
```

ファイルの内容をメモリにリストアします。生バイナリ以外の場合は自動でフォーマットを判定します。生バイナリを指定する場合は binary の指定が必要です。

bias がゼロでない場合、値がファイルに含まれるアドレスに追加されます。