

[スクリプト言語 Pawn とは]

、

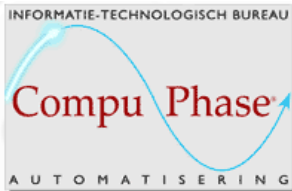
特徴

1. C言語によく似たスクリプト言語
2. インタープリタではなく、コンパイルしてバイトコードを生成して、それをランタイムライブラリで実行する
3. 他のスクリプト言語に比べてると、実行速度が非常に速く、メモリ使用量も少ない
4. スクリプトがスタック領域をあらかじめ確保して、その中でメモリのアロケートを行う

主なゲームの採用例としては、HalfLife2,GrandTheftAuto 等

こちらのサイトに豊富なドキュメントが用意されています。

<http://www.compuphase.com/pawn/pawn.htm>



gflib では

ソースとラッパー
gflib_cpp/opensource/pawn

◆スクリプトソース

ソースファイル拡張子は .p ヘッダファイル拡張子は .inc

生成されるバイトコードファイル拡張子は .amx

```
hello.p

#include <console> // like stdio
main()
{
    /* おなじみのやつ */
    printf("Hello world¥n");
}
```

これを見れば、どんなに
C言語に近いかが分かる
と思います！

- コメントキーワードは // /**/ でC++と同じ
- 文末の ; は必要ありませんが、あっても問題はありません。
- console は stdio のような Pawn のライブラリです。
- メイン関数がエントリポイントになりますが、ホスト側から関数単位で呼び出すことも可能です。

◆ホスト側のプログラム

次はこれを実行させるホスト側のプログラムの例です。

```
gfl::pawn::Pawn pawn;           // ラッパークラス  
pawn.Load( "kimura/hello.amx", heap ); // ファイルをロード  
pawn.Update();                  // main() の実行
```

- この例の場合は、内部的にスマートポインタを使用していますので pawn クラスが破棄されるときにロードしたメモリも自動的に破棄されます。
- 通常のゲームで使用する場合は初期化時に Load し、
毎フレーム Update をコールするようになると思います。
- Pawnのラッパークラスは 3DS,PC上の同じコードで動作可能になっています。

◆変数の宣言

- キャラや整数等の型の区別はありません。
ポインタ、構造体やクラスはありません。
配列は 3 次元まで可能です。
- 関数内での new はローカル変数、関数外ならグローバル変数になります。
static 宣言されるとスタティック変数になり値が保存されます。
public 宣言されるとホスト側から参照、変更が可能になります。
const 宣言は定数になります。
- sizeof 演算子がありますが、データのサイズではなく要素の数になります。
テーブル内のアイテム数を求めるのに
C言語だと

```
s32 tbl[] = { 0,1,2 };  
sizeof(tbl)/sizeof(s32)
```

ですが、スクリプトでは

```
new tbl[] = { 0,1,2 };  
sizeof(tbl)
```

になります。

- 浮動小数を使用する場合は例外的に ヘッダが必要です

```
#include <rational>      // このヘッダが必要  
new Rational:ang = 2.9;
```

- このように明示的な宣言が必要になります。
- 数値の最後に f は付けられません。

◆関数の宣言

- スクリプト内部での関数宣言は必要ありません。
関数自体が呼ぶ側の上でも下でもかまいません。
引数と取ることが可能で、返り値もあります。
- ホスト側の関数を使用する場合は宣言が必要になります。
スクリプト側の宣言方法は以下になります

```
native GameSideFunction( val, float_val );
```

- ホスト側はスタティック関数として宣言します

```
static cell AMX_NATIVE_CALL GameSideFunction( AMX* amx, const cell* ptr )
{
    // AMX引数は pawn スクリプト本体のポインタになります
    // 引数は添字 1 から始まります

    s32 val = ptr[1];

    // 浮動小数を受け取る場合はマクロが必要になります
    amx_ctof( ptr[2] );

    // 返り値はスクリプト側でそのまま受け取れます
    return val + 1;
}
```

- 次に関数をテーブルに登録します

```
static const GFL_PAWN_FUNC_TABLE s_FuncTable[] = {
    GFL_PAWN_FUNC( GameSideFunction )
    GFL_PAWN_FUNC_END
};

RegisterPawnFunction( s_FuncTable );
```

この登録はスクリプトのロード後に pawn クラス内で行ってください。

◆インクルードファイル

- enum, #define が使用可能ですので、スクリプトとホストで共通のインクルードファイルが作成出来ます。

```
common.inc

// モード
enum {
    MODE_INIT,
    MODE_MAIN,
    MODE_END
};
```

```
// 初期化サイズ
#define INIT_SIZE 10
```

のようなファイルなら、定数の不一致などの障害を防ぐことが可能です。

◆関数の参照渡し（１）

- 関数にポインタ渡しはありませんが、参照渡しは可能です。

```
swap(&a, &b)
{
  new temp = b;
  b = a;
  a = temp;
}
```

- デフォルト引数を持たせることも出来ます。

```
func(a,b=0)
{
  return a+b;
}
```

- 可変引数も可能です。

```
sum(...)
{
  new result = 0;
  for (new i = 0; i < numargs(); ++i)
    result += getarg(i);
  return result;
}

new val = sum ( 1,2,3,4,5);
```


◆関数の参照渡し（２）

- ホスト側から呼ぶ関数には public 宣言が必要です。

```
public CalledByHost()
```

@CalledByHost() のような書き方も出来ます。

- スクリプトをライブラリ化したい場合
関数に stock 宣言をつけると参照のない時には関数のインスタンスを作成しません。
- 再帰はサポートされていません。

◆プリプロセッサ

プリプロセッサとして有効なのは

```
#assert #define #else #elseif #endif #endinput #error #file #if #include #line  
#pragma #section #tryinclude #undef
```

define はマクロとして使用可能です。

```
#define min(%1,%2) ((%1) < (%2) ? (%1) : (%2))
```

引数は %数字 で表すところがC言語との違いです。
数字部分は 1 から 9 と 0 の 10 個までが可能です。
C言語と同じように # で文字列化が出来ます。

```
#define a(%1) "Value is #%1 ¥n"  
  
print a(100);
```

ほとんどが、C言語と同じ使用方法です。

```
assert break case continue default do else exit for goto if return sleep  
state switch while
```

ただし、switch は注意が必要です。

```
switch(step){  
  case 1 ... 9:      // GCC拡張と同じように、1...9は1から9までの数値になります。  
    printf("step %d¥n",step);  
  case 10:  
    printf("just 10¥n");  
  default:  
    printf("more than 10¥n");  
}
```

- switch 分の中には break がありません。
- case のあとの 1 命令を実行します。
- break は for while を抜けるのに使用します。

◆オペレーター

これもほぼC言語と同じです。

```
+ - * / % ++ -- ~ >> >>> << & | ^ = += -= *= /= %= >>= >>>= <<= &=
|= ^= == != < <= > >= ! || && [] {} () ?: : defined sizeof state tagof char
```

>> と >>= は符号付きシフト、>>> と >>>= は符号なしシフトです。

◆コア関数とコンソール関数

コア関数

```
clamp funcidx getarg heapSPACE max min numargs random setarg swapchars  
tolower toupper
```

コンソール関数

#include <console>が必要です。

```
getchar getstring getvalue print printf clrscr clreol gotoxy setattr
```