題名 :#pack の呪い

(まえがき). 作品の詳細は後方に説明致します。この作品は、ゲームプレイユーザーではなく、HSP コンテストへ同じく参加しているプログラマー兼コードライターである人達、いわば私と間接的な同業者と言っても過言ではない人たちに向けて作った作品です。したがって、以下に執筆する本文は私と同じ間違いを犯し、ソースコードを直すことへ一苦労した人に向けてのせめてもの餞をしたいという思いで骨身を削って作った作品です。私を苦しめた、その極悪非道ともいえる命令、その名は「#pack」と名付けられていました……。

1. 動機

私が最初に「#pack」へ目が留まったのは、作品が完成したときでした。遡ること 1 年程前、実際に友人に完成した作品を見せようとしたら、その友人は「ゲームの中で見るからこそ、華が咲くと言えるべき画像」を先に見てからゲームを始めやがったのです。そのせいで、これから起こり得た最高のエンターテイメントを全部踏みにじったままゲームを強行し、終始つまらなそうな顔をして私のゲームをプレイしていました。結果が分かってしまうゲーム程、つまらないものってありませんよね。

そうして私がたまたま通りかかった「#pack」というプリプロセッサ命令は当時、革命的な物のように思えました。なんといっても画像や音声ファイルを packfile に追加をし、ゲームをプレイするユーザーに対して、内容を非公開にしたままゲームを進行させることはある意味私が求めていたものに合致していたからです。昔は、PACKFILE 一覧のダイアログからファイルの追加や削除を行っていたことも、その時知りました。今こうして振り返ってみると、PACKFILE に追加し忘れていたファイルが 1 つでもあると最初からリスタート、という途方もない作業は、すごく大変だったんだなと同情してしまいます。その分「#pack」命令はファイルの抜け落ちがあったとしても内容はスクリプトの方へ保存されてくれるため、少しは救済だったのかもしれません。

しかし、そこに待ち受けていた地獄の門は着実に私の怒りを奮い立たせていました。何よりも痛ましかった最初の始まりは、「2 バイト文字禁止」でした。テキストファイルは、こちらから見てもすぐわかるように、私は日本語で管理をしていました。HSP をしているプログラミングの有識な読者様にはもう分かっていると思いますが、日本語はがっつり 2 バイト文字です。全てローマ字などで、1 バイトの文字に統一し直しました。

ここで終わったのなら、本当はハッピーエンドだったんです。ハッピーエンドをここで壊したやつがいるから、私が今こうして「Converter Station.pdf」を書いているのです。そいつの名は……

やはり「#pack」でした。

実行ファイルが作成できたとしても、いざ開くと、内部エラーを引き起こしやがりましたこいつは。賢明な読者の皆様ならご存じだと思いますが、「#pack」においてパスを「//」等で繋ぐと、どうやらダメなようです。私がこれに気づくことができたのは、有識者の皆さんに教えて頂いたお陰であります。本当に感謝しています。しかし、この件だけで約 3 週間程の時間を要しました。この時は心から本当に勘弁してくれ……と頭を垂れ流していました。

ここで終わっていたならば、本当にハッピーエンドの範疇で終わっていたのです。しかし、ここまでの流れから推測してもらえば分かるように当然ハッピーエンドを悉くブチ壊してやってくるものがいるのです。ご存じの通りまた「#pack」がやってきました。度重なるエラーの画面と対面し、20 秒近く顔面がフリーズしていたと思います。その後、思考放棄をして 3 か月近く月日が経過していきました。

丁度「#pack」命令を忘れていた頃、PC 画面の前に偶然通りかかり、改めてエラーの発作を反射的に感じていました。その時、私が今一度 PC と向かい始め、確認したことは、「celloadの 1 つ 1 つのファイルを抜き差しする」という地道な作業でした。HSP 上今世紀最大に疲れたと思います。ですが、こんな地味な作業でさえも確かに得られたものは存在していました。それは「パスで通しているファイルが全てエラーになったということ」です。これは「¥¥」でパスを通したからもういいだろうと思っていましたが、ところがどっこい、裏でもう 1 つ落とし穴があったのです。なんと packfile 内ではディレクトリのような階層表現をすることができないのです。(これは流石に分からなかったので調べました。)まさか、パスの通し方で問題があったのに、次は新たに「パスを通すこと自体」がエラーと言われ、心底「#pack」が嫌いになりました。今まで頑張って打ってきた「celload の『¥¥』」を消し、実行ファイルを作成しなければならなかったのです。

幸運にも字数の多いファイルを取り扱っていなかったためか、「15 文字を超えるファイル名によるエラー」は当たることがありませんでした。また、音声ファイルでは「mp3」形式を取り扱えないということだったので全て「wav」拡張子に揃えました。「#pack」命令 1 つとファイルを書き込むだけで簡単に終わると思っていた命令だったのですが、実に 4 か月以上もの間振り回されていたという事実に驚きを隠せませんでした。ここまで「#pack」と犬猿の仲を維持できたのも今となっては誇らしいです。

しかも、その後に及んでも「#pack」の負のスパイラルは強さを日に日に増して行き、celload 命令の階層表現をやめてしまったことで今度は逆に「F5」による実行ができなくなってしまいました。「#pack」はどこまでも私のことを呪ってくるのです。このエラーの一連の流れを私は、個人的に「#pack の呪い」と呼んでいます。

軽弾みな気持ちで使い始めた「#pack」という命令は、リファレンスのみを参照すると簡単のように見えますが上記のような、パス絡みの問題になると大変なことになります。したがって、こんな単純な落とし穴で躓いている人がいるとしたならば、何かしらの助けになってあげたいと、心から思うようになり、このときから、簡単に「#pack」命令と階層表現を無くした「celload」命令を自動表記してくれる、「ソースコードの改変を行うソフト」を作

ることで、「#pack」命令において生じた被害者を撲滅しようと思いました。「#pack」命令に関して同じ大変さを味わった人が少しでも報われるよう精一杯作ったので、「#pack」命令をこれから実装するよ、という人はぜひ本作品を試して欲しいです。

2. 実装した機能

ただ単純に「celload」を「#pack」という文字列に置換するだけであるならば、HSP タブについている「文字列の置換(R)」だけで十分です。しかし、私は 1 章の動機で示したものの他に以下のようなものも実装をしたかったのです。今回はそれを順に追って説明していきます。

・(1) ファイルを(「Ctrl+A」で全選択して)ドラッグ&ドロップをすることで簡単に「#pack」置換までしてくれるショートカット機能。(ディレクトリまで指定できる機能付き)

```
// D&D を行ったファイル
picturea.png
pictureb.png
picture.png
```

→

```
「変化後」
#pack “picturea.png”
#pack “pictureb.png”
#pack “picturec.png”
```

＜ドラッグ&ドロップ時の深度とは＞
・メッセージ用のボックスに整数値を指定することで以下の役割を担う。
ボックス内の数値：
「1 以上の値の場合」 → ファイルから指定された数値分上のディレクトリも考慮して表示

「上記以外の値や文字列の場合」 → ディレクトリを考慮しない形でファイル表示

・(2) 「#epack」か「#pack」かを選択できる置換を行う。

```
picturea.png
```

→

```
「変化後」
#pack “picturea.png”
```

※自由に選ぶことが可能

```
「変化後」
#epack “picturea.png”
```

・(3) 「#pack」で有効な拡張子か判別してから、「#pack」置換をしてくれる機能。
（「拡張子が間違っているもの 、 文字数が 15 文字を超すもの 、 2 バイト文字を使用しているもの & スペースが使われているもの」）
また、「/」でのディレクトリ間移動を「¥¥」に置換し、「#pack」命令で扱えるようにする。

```
celload “picturea.png”
addxfile “pictureb.x”
picload “picture.hsp”
cellooad “abc//def.png”
celload “abcdefghijkl.png”
celload “画像.png”
celload “abc def.jpg”
```

「変化後」

```
#pack “picturea.png”
#pack “picture.x”
#pack “abc¥¥def”
```

・拡張子が間違っている →
・文字数が 15 字を上回っている →
・2 バイト文字を使用している →
・スペースが使われている →

「エラー」

```
picload “picture.hsp”
celload “abcdefghijkl.png”
celload “画像.png”
celload “abc def.jpg”
```

＜・#pack で置換できるよう設定した拡張子一覧＞
（※start.ax ファイルは自動的にパックされるので指定しない）
・「BMP、GIF、JPEG、PNG」の画像ファイル
・「WAVE」形式の音楽ファイル
・「TXT」形式のテキストファイル
・「X」形式の 3D モデル用のファイル

・(4) mesbox による直接編集できるオブジェクトを用いて、そのまま置換するだけでなく「元の形」と「変化後」をダブルボックスで確認できるような置換を行う。
（「元の形」ではディレクトリが反映されなくなるため注意。）

```
celload “a//picture.png”,1
picturea.png   // 編集で追加
picture.b.jpg   // 編集で追加
```

「変化後」

```
#pack “a¥¥picture.png”
#pack “picturea.png”
#pack “pictureb.jpg”
```

※画像や音楽ファイル等の読み込み命令が探知されない場合、元の形は反映されない形で表示を行う。

「元の形」

```
celload “picture.png”,1 // 階層表現なし
// picturea.png は反映されない
// picture.b.jpg は反映されない
```

・(5) 「:」の考慮ができる「#pack」の置換を行う。結果の表示では高速処理を目的として、「:」を排除しているため注意。

```
a.png : b.png
celload “dir//c.png”,1 : celload “d あ e.png”,2
```

「変化後」

```
#pack “a.png”
#pack “b.png”
#pack “dir¥¥c.png”
```

「元の形」

```
celload “c.png”,1
```

「エラー」

```
celload “d あ e.png”,2
```

※以下のような変則的なファイル指定はドラッグ&ドロップしたファイルと酷似するためエラーと見なされる。(celload などの読み込みを行う命令は「”」が 2 つで囲まれた部分のファイル名で変換を行う。)

```
repeat 3
        celload “pic(”+cnt+”).png”,1+cnt
loop
s = “a.png”
celload s,1
t = “b.png”
t
```

「エラー」

```
celload “pic(”+cnt+”).png”,1+cnt
celload s,1
```

・「#pack」へ変換されるときに作られる種類訳は 3 つある。

・変化後 … 読み込み命令のファイルや D&D をしたファイルを「#pack」に変換したデータの集まり。階層まで書き込み「/」で繋げていたパスを自動的に「¥」に変換を行って出力する。

・元の形 … 読み込み命令のファイルのみ反映される。階層表現を全て取り払った形で、新規に読み込み命令を書き込み、出力を行う。

・エラー … 変換ミスが起きたファイルのみ反映される。字数過多や 2 バイト表記などの要因で扱えないものを抽出して書き込みを行う。

・読み込み命令に対応する拡張子一覧

命令：「bload」　→「.txt　、　その他の拡張子」を持つファイル

命令：「noteload」→「.txt」を持つファイル

命令：「picload 、 celload 、 loadtoon 、 texload 、 texload2 、 imgload」
→「.bmp 、 .jpg 、 .gif 、 .png」を持つファイル

命令：「mmload 、 dmmload」→「.wav」を持つファイル

命令：「addxfile 、 addxanim」→「.x」を持つファイル

3. 機能の拡張

「bload」命令などでは「.txt」以外にもユーザーが指定した拡張子を扱えます。しかし、「#pack」命令を使用できるかどうかは HSP に依存しているため注意が必要です。今回、HSP が「#pack」命令を拡張することを見越し、「#pack」に変換できる拡張子をユーザーが選べるようにしました。以下の手順によって、同梱されている「Converter Station.hsp」のスクリプトを改竄して、機能を拡張させてください。

1.命令の拡張を行う

・以下の「ordernum」(l.58)の値を増やし、命令の個数を拡張してください。その後「種類」というコメントに従い、それに該当する場所へ新規命令を書き込んでください。

```
56
57 extensionkind = 5 //  「#pack」に変換する型の種類指定
58 ordernum = 6 //  「#pack」に変換する命令の個数指定
59 orderliteral = 50 // 命令文の文字数指定
60 sdim order,orderliteral,extensionkind,ordernum // 命令文拡張子分確保
61
62 /*  種類 : 1 …「画像に関する読み込みに対応する命令」
63     種類 : 2 …「音楽に関する読み込みに対応する命令」
64     種類 : 3 …「3Dモデルに関する読み込みに対応する命令」
65     種類 : 4 …「テキストに関する読み込みに対応する命令」
66     種類 : 5 …「種類 : 1～4 のいずれにも該当しない命令」   */
67
68 extcount = 0   : extcnt = 0 // 拡張子の種類の判別を付けるためのカウン
69 order(extcount,extcnt) = "picload" : extcnt++ : order(extcount,extcnt
70 extcount++ : extcnt =0 // 拡張子の種類を移動して、カウント変数をリセ
71 order(extcount,extcnt) = "mmload" : extcnt++ : order(extcount,extcnt)
72 extcount++ : extcnt =0 // 拡張子の種類を移動して、カウント変数をリセ
73 order(extcount,extcnt) = "addxfile" : extcnt++ : order(extcount,extcn
74 extcount++ : extcnt =0 // 拡張子の種類を移動して、カウント変数をリセ
75 order(extcount,extcnt) = "noteload" //  「#pack」に関する命令定義
```

2.拡張子の拡張を行う

・以下の「extensionnum」(l.80)の値を増やし、拡張子の扱える幅を拡張してください。その後「種類」というコメントに従い、それに該当する場所へ新しい拡張子を書き込んでください。

```
80 extensionnum = 4 // 「#pack」に変換する拡張子の個数指定
81 extensionliteral = 10 // 拡張子の文字数指定
82 sdim ext,extensionliteral,extensionkind,extensionnum // 拡張子名拡張子分確保
83
84 /*  種類 : 1 …「画像に関する読み込みに対応するファイル拡張子」
85     種類 : 2 …「音楽に関する読み込みに対応するファイル拡張子」
86     種類 : 3 …「3Dモデルに関する読み込みに対応するファイル拡張子」
87     種類 : 4 …「テキストに関する読み込みに対応するファイル拡張子」
88     種類 : 5 …「種類 : 1～4 のいずれにも該当しないファイル拡張子」  */
89
90 extcount = 0  : extcnt = 0 // 拡張子の種類の判別を付けるためのカウント変数
91 ext(extcount,extcnt) = ".bmp" : extcnt++ : ext(extcount,extcnt) = ".gif" : extcnt++
92 extcount++ : extcnt =0 // 拡張子の種類を移動して、カウント変数をリセット
93 ext(extcount,extcnt) = ".wav" // 「#pack」に変換する拡張子名定義
94 extcount++ : extcnt =0 // 拡張子の種類を移動して、カウント変数をリセット
95 ext(extcount,extcnt) = ".x" // 「#pack」に変換する拡張子名定義
```

該当する場所とは、真下に記述されている、文字の羅列のことです。「extcount」という変数で種類の場所を操作し、「extcnt」という変数でその種類の中で機能する命令・拡張子を指定しています。上の段落から順当に「種類 1・種類 2・種類 3・種類 4・種類 5」と並んでいますので、その行に合わせて語尾に「extcnt++ : order(extcount,extcnt)=”新規命令”」、もしくは「extcnt++ : ext(extcount,extcnt)=”新規拡張子”」を追加してください。

拡張したからと言って、必ずしも「#pack」が実行されるとは限りません。あくまでも、このソフトは、「読み込み命令」→「#pack」の書式変換をするためのものであり、「#pack」命令自体とは何の関係性もありません。命令を拡張するときは、必ず「#pack」命令が通るか確認をしてから、拡張するようにお願いします。ドラッグ&ドロップについても拡張した命令・拡張子が反映されるため注意が必要です。

4. ドラッグ&ドロップについて

ドラッグ&ドロップでは複数のファイルをドラッグすることが可能になっています。ディレクトリ内にあるファイルを全て「#pack」形式に変換したい場合には、「Ctrl+A」で全選択を行い、メッセージボックスの枠内に挿入してください。枠から外れたりした場合は読み込まれない可能性があります。また、部分的に挿入したり、複数回同じものを挿入することも可能です。(実行結果では、同じファイル名は「#pack」命令で反映されませんが、ディレクトリが違う同名ファイルは読み込まれます。しかし、読み込み時は階層表現がなくなるため、「#pack」された同一ファイル名はできるだけ作らないようにすることを推奨します。)

5. お勧め変換方法

自分の作品が完成した場合、予め「#pack」用の hsp ファイルを作っておきます。(バックアップを取っていた方が「F5」で実行する用と、実行ファイルを作成する用で分けられるため。)そうしたら、その「#pack」用の hsp ファイルを「Ctrl+A」で全選択を行い、そのまま「Converter Station のメッセージボックス内」にコピー&ペーストを行ってください。そして「変換実行」というボタンを押していただければ、ファイルに関して、エラーが出た文と成功した文が出ると思いますので、エラーに入っている文の字数や文字に変更を加えて、再度エラーが起こらないようにします。(カウント変数や文字変数を用いた読み込み命令はエラーとなるので無視して構いません。)エラーの欄が空文字になれば「#pack」が全てに適応できるということになりますので、そのまま、「変換後」の文を全選択し、hsp スクリプトの先頭に添付してください。また、ディレクトリに階層がある場合は、「元の形」からコピー&ペーストして上書きをし、完了したら「Ctrl+F9」によって実行ファイルを作成すれば完成となります。(必ずバックアップ用の hsp ファイルを作るようにしてください。また、ファイル名を変える際にも十分注意してください。)

6. 総括

「#pack」は手動で直そうとすると、読み込み命令を文字列の置換で「#pack」に直すことが一般的だと思っていますが、それではパラメータを削除することに大変な手間がかかりますので、今回このようなソフトを作らせていただきました。少しでもこのソフトを通じて作業が捗れば幸いです。また、それと同時に「#pack」へ書式を変えるということは元の形に戻すこともまた大変な作業になるはずです。したがって、ミスが偶発的に起こってもいいように、「バックアップを取る」ということを何度も推奨しておきます。このソフトも何かしらの致命的なミスがあれば、その都度、修正と向上に努めてまいります。そして、「#pack」の良さについてもたくさん知ってもらいたいです。みなさんが「#pack」の呪いの瘴気に触れることなく突っ切れることを心より願っています。