犯しやすいミスの指摘からデバッガの使い方まで

初心者からのステップアップ

岸 哲夫

この章では初心者からステップアップするためのノウハウや，オープン・ソースをどのように利用するか，デバッガの使い方などを説明する．深く学ぶ際には，ネットや書籍を利用するという読者も多いことだろう．その際の前知識としてほしい．（筆者）

初心者でもベテランでもミスをする

●「¥」に注意

通常のLinux環境では漢字コードにShift-JISを使ったりはしません．したがって問題は起きませんが，事情があってShift-JIS環境で動かしている場合や，デフォルト設定のCygwinではShift-JISを使用します．そのほかの漢字コードなら問題にならないことですが，「¥」(0x5c)を使用している漢字の扱いには注意しなくてはなりません．また，この例のように文字列中に「¥」が必要なために使用しているソースも注意しなくてはなりません．Shift-JISを使わなければ何も問題は起きません．

リスト1に文字列中に¥を使ったソース・リストを，**図1**に実行した状態を示します．

やはり「¥」が入った文字列は認められません．EUCを使う環境で試したところ，「\」(バックスラッシュ)になってしまいました．回避するには**リスト2**のように，全角¥を使うのがよいのではないかと思います．**図2**は実行結果ですが，多少は不恰好になってしまうことは否めません．

¥がバックスラッシュになってしまう原因は，Cygwinで使用しているターミナルのフォントが半角の¥を\に勝手に書き換えてしまうことである場合もあります．英文字フォントを変えて，**リスト3**のソース・リストのようにすれば，問題ない場合もあります(**図3**)．

文字コードに関しては①などの丸囲み数字が環境によって読

リスト1　文字列中に¥を使ったソース・リスト(test74.c)

```
/*
 *¥の問題
 */
#include <stdio.h>
#define  kingaku   "¥1000"
int main(void)
{

        printf("金額=%s¥n",kingaku);
    return 0;
}
```

```
$ ./test74
金額=@0
```

図1　test74.cをコンパイルして実行した結果

リスト2　文字列中に全角¥を使ったソース(test75.c)

```
/*
 *¥の問題
 */
#include <stdio.h>
#define  kingaku   "￥1000"
int main(void)
{

        printf("金額=%s¥n",kingaku);
    return 0;
}
```

```
$ ./test75
金額=￥1000
```

図2　test75.cをコンパイルして実行した結果

リスト3　文字列で¥を使ったソース・リスト(test76.c)

```
/*
 *¥の問題
 */
#include <stdio.h>
#define  kingaku   "¥¥1000"
int main(void)
{

        printf("金額=%s¥n",kingaku);
    return 0;
}
```

```
$ ./test76
金額=¥1000
```

図3　test76.cをコンパイルして実行した結果

KEYWORD ── ¥，shift-JIS，文字列ポインタ，変数の初期化，スタティック変数，char型，int型，引き数，最適化オプション，共通部分式，定数の畳み込み，ループ不変式，2のべき乗の乗除算

めなかったり，NEC特殊文字やIBM拡張文字の問題もあります．対象の環境に注意しましょう．

リスト4は0x5cをコード内に含む漢字を使ったソース・リストです．**図4**の結果を見てわかるように，改行のための「¥n」が壊れています．

リスト5のように，単純に「¥」をもう1個余分につけるだけでうまくいきました(**図5**)．この手の漢字をチェックするツールがMS-DOS時代の昔からありますが，これを利用してみるとよいと思います．

つまり「¥」がC言語上やOS上で意味をもって使われるので，それを考慮していないShift-JISの漢字コードを使っているとき，漢字コードの2バイト目が0x5cになる漢字が出現すると問題が起きるわけです．

● 文字列ポインタ全般

リスト6のソース・リストは「文字列ポインタ」の配列です．配列の要素数が4だと思ってプログラムしていますが，ご覧のとおり「,」が抜けていて，要素数3でプログラムされています．

char **chkは「ポインタの指す場所」つまり「ポインタのポインタ」なので「*」が2個付いています．

図6に示すように，結果は3個の要素を出力して終わってい

ます．これも「,」がないためにバグの元となってしまいます．

リスト7は4個の文字列のポインタを配列に保存しています(**図7**)．やってることは違いますが，**リスト6**と**リスト7**は同じ結果になります．

しかし環境によってはchar *pt1 ～ *pt4は書き換えできない領域に確保されます．このような処理を行うと*chkが指す領域が書き換えできるのかできないのか判別がつかなくなります．このような処理を行うときは大いに注意すべきです．この環境で動作しても，別の環境では動作しないかもしれません．

Cygwin上でx86向けに構築されたgccでは，どちらの文字列も書き換え可能なアセンブラ・ソースを生成します(**図8**，**図9**)．

とくにどちらの文字列もtextセクションに置かれていません．ただし，ほかの環境ではどうなるかわからないので，書き換える可能性がある文字列はchar buf[]="aaaaa";などのように明確に配列にしておいたほうが無難です．

ちなみにtextセクション(.text)にはプログラム本体や初期化されて変更する必要のないデータを置きます．

dataセクション(.data)は初期化が必要なデータを置きます．dataセクションのデータは実行時に書き換えることができます．

bssセクションは実行時にメモリを割り当てる領域です．実

リスト4　0x5cをコード内に含む漢字を使ったソース・リスト(test77.c)

```
/*
 *¥を含む漢字の問題点
 */
#include <stdio.h>
int main(void)
{

        printf("表¥n");
    return 0;
}
```

```
$ ./test77
表n
```

図4　test77.cをコンパイルして実行した結果

リスト5　0x5cをコード内に含む漢字を正しく表示するソース・リスト(test78.c)

```
/*
 *¥を含む漢字の問題点
 */
#include <stdio.h>
int main(void)
{

        printf("表¥¥n");
    return 0;
}
```

```
$ ./test78
表
```

図5　test78.cをコンパイルして実行した結果

リスト6　文字列ポインタを使ったソース・リスト(test79.c)

```
/*
 *文字列ポインタ
 */
#include <stdio.h>
int main(void)
{
    char *buf[]    =    {    "123dfghtg",
                             "456asdfghaaa"
                             //「,」を付け忘れているので配列の要素数4だと思っても実際は3
                             "789lkjljuyiydssssss",
                             "ABCetrtrytrtrtrtrtr",
                             '¥0'                    };
    char **chk;                          //各文字列のポインタが保持しているアドレスを格納
    int       ix   =    0;
    for (chk = buf; *chk != NULL; chk++)
    {
         printf("%s¥n",*chk);
    }
    return 0;
}
```

```
$ ./test79
123dfghtg
456asdfghaaa789lkjljuyiydssssss
ABCetrtrytrtrtrtrtr
```

図6　test79.cをコンパイルして実行した結果