本誌正式名称は「シェルスクリプトマガジン vol.26」です

月刊 世界で唯一の

Shell Script Is The Supreme Glue Technology
FOR THE SOPHISTICATED SHELL SCRIPTERS

シェルスクリプトマガジン

よりぬき版

2015 6 June Vol.26 ¥500

TechLION再録
Javaの20年、テープバックアップ、災害コミュニケーション…ITの20年を語る！
櫻庭祐一×堂前清隆×松本直人

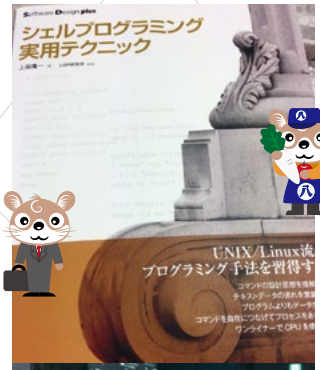
Haskell版Open usp Tukubai 完成させるぞ企画
Haskellでやってはいかんのか？

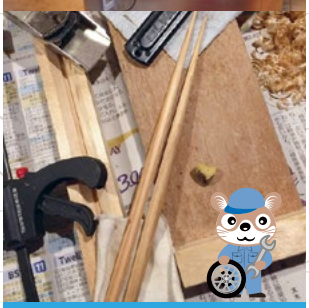
IPv6新時代を体感しよう！
ユニケージ開発手法 コードレビュー
中小企業診断士が解説する、超実践的な会話術！
円滑コミュニケーションが世界を救う！

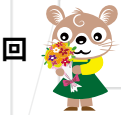
特集
中小企業手作りIT化奮戦記

C言語のススメ

CONTENTS







ちんじゅうちゃん

本誌名称について
vol.21より「USPMAGAZINE」から「シェルスクリプトマガジン」に、変更となりました。バックナンバーは「USP MAGAZINE」としてお求めいただけます。
本誌正式名称は「シェルスクリプトマガジンvol.26」となります。

世界で唯一の
月刊シェルスクリプトマガジン
2015 June vol.26

風待月

今月の特集は好評連載中の「中小企業手作りIT化奮戦記」！

触った経験が無い人へ「C言語のススメ」

人気連載

手作り中小企業IT化奮戦記

writer 菅 雄一

C言語のススメ

システム管理者といっても仕事内容は様々だ。
シェルを駆使してUNIXやLinux管理のツールを作成している人もいれば、PHPやRubyのWeb系プログラマーだったり、汎用機のRPG/400やCOBOLを書く人もいるだろう。

そしてC言語を触った経験すらない人、プログラムを全く書かないシステム管理者がいても不思議ではない。正直なところ私自身、C言語は初歩的なプログラムしか書く事ができない上、C言語で書かれたオープンソースを解読するだけの読解力もない。だが、私の経験ではC言語の知識は多少持っておいた方が、様々なシステムやOSの動きを知る上で理解しやすいため、役に立つ言語だと明言できる。

なぜなら多くのシステムやOSが、このC言語で書かれているため、C言語の問題は、システムの問題に直結するからだ。そしてC言語を書いたデニス・リッチーは、「Cをもっぱら UNIXのシステムを記述するための言語とみていた。(中略) Cは十分に低いレベルにありますから、システム設計の約束事を忠実に守っていけばなんでもできる(※)」と言っている。(※出典:「C MAGAZINE 1989年 10月号」)これはUNIX愛好家であるシェルスクリプトマガジン読者にとっても聞き逃せない言葉だろう。

そこで今回はC言語を触った経験のない人向けに、C言語の概略を説明しながら、C言語の学習で得られた知識が、システム管理に役立つ話を書くことにした。

■プログラミング言語とは

プログラミング言語は、平たくというとコンピューターに命令して処理を行うための言語だ。
「1+1を計算しなさい」や「Helloを表示せよ」といった命令だ。

コンピューターは「0」か「1」の二進数の並びの命令を理解する。コンピューターが理解できる言語を機械語という。機械語で16進数表現されているものは、「0」「1」の並びだと見えにくいため、16進数に置き換えているのだ。

だが、機械語だと人間が扱うには困難だ。そこで、人間が読み書きしやすい形式の命令文になったのがC言語だ。他にもBASIC、FORTRAN、Java、PHPなどがある。人間がわかりやすい言語を「高水準言語」または「高級言語」という。諸説あるようだが「0」「1」の機械語は原始的なので「低水準言語」と呼び、色々な命令表現があるC言語やJavaなどの事を高水準言語と呼んでいるようだ。

高水準言語だと人間は理解できても、コンピューターには理解できないため、コンピューターにわかる機械語に翻訳する。翻訳作業の事をコンパイルと呼び、コンパイルをするソフトのことをコンパイラーという。コンピューターと対話するための言語であるプログラミング言語は、毎年のように生まれている。

■C言語は英語

C言語を勉強しはじめた頃は、闇雲にC言語の命令を覚えていった。しかし、丸暗記だったため、すぐに忘れてしまう。世界のプログラマがこの無味乾燥な命令を覚えられるのが不思議で仕方なかった。

だが、だいぶ後になって、英語の略語であることに気づいた。例えば、strcpy()命令は「string copy」の略で「文字列の複製」という意味でそのまんまだ。ファイルを開くfopen()命令は、「file open」の略だ。

C言語はアメリカで開発された言語のため、英語を使うアメリカ人がわかりやすい仕様になるのは当然だ。

中学・高校で英語を学んでいる我々にとっても比較的、英語は馴染みやすい。もし、ドイツやフランスで開発され、命令文がドイツ語やフランス語の略語なら、命令文は無味乾燥で丸暗記になり、非常にとっつきにくくなっていただろう。

■C言語のプログラム

C言語の入門書に出てくる簡単なプログラムの１つに、「Hello World」を表示させるものがある。

```
#include <stdio.h>

int main(void)
{
  printf("Hello World");
  return(0);
}
```

#include <stdio.h>は、stdio.hというファイルを取り込む命令だ。includeは「取り込む」なので、そのまんまだ。拡張子「h」のファイルはヘッダーファイルと呼ばれ、関数の型式が記述されている。プログラム中で使う関数を宣言する際、該当する関数の宣言が記述されているヘッダーファイルを取り込む必要があるのだ。(他にも変数の定義、構造体の定義もある)

stdio.hのファイル名は「standard input output」の略で、「標準入出力」に関する関数群の型式が記述されている。(C言語の入門書のサンプルプログラムに載っているヘッダーファイルは、おまじないだと思っても良いかもしれない)

■C言語は関数の集合体

C言語の命令は関数形式が多い。数学の関数と全く同じものだ。関数とは入力値を変換する装置だ。入力値 (x) の値が決まると、出力値 (y) の値が決まる。そのため数学では、x の値を関数 f に代入して y の値が出てくる式を y=f(x) と表現している。

数学が苦手な人には難しく思えるが、身近な所に関数はある。具体例は自動販売機だ。アイスコーヒーのボタンを押すと、アイスコーヒーが出てくる。ホットコーヒーは出てこない。押すボタン（入力値）によって、出てくる飲み物（出力値）が決まっている。飲み物=自販機（ボタン）という関数なのだ。

C言語も同様で、printf(“Hello World”); の命令は、入力値「Hello World」が、関数 printf() によって、出力値として、画面に「Hello World」と表示される。C言語の場合、入力値とは言わず、引数という。この場合、引数は文字列「Hello World」になるのだ。

ところで、プログラムの本体になる main() も関数だ。関数の宣言と、関数の中身を記述している。そしてmain関数の宣言の引数部分にある「void」は「空」とか「空虚」という意味で、引数がない事を意味する。C言語の関数で引数がない場合にvoid が使われる。main() は関数なので出力値がある。宣言部分にint があるが、int は integer の略で「整数」の意味だ。int main() は、main()関数の返り値は整数という意味だ。return()命令を使って出力値「0」を出している。「0」は正常終了を意味している。

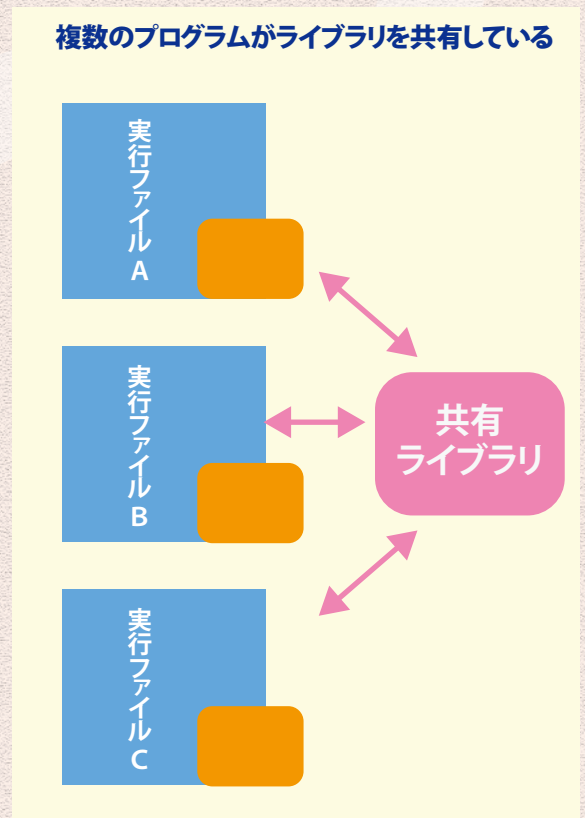
■ライブラリ

C言語には多くの関数が用意されている。それらはライブラリと呼ばれるファイルに格納されている。libraryなので、まさに「プログラムの部品（関数）の図書館」という感じだ。先ほど触れたヘッダーファイルは、関数の宣言(関数名と引数の数)の記述であって、関数の中身は記述されていないのだ。ところでライブラリには2種類ある。静的ライブラリと共有ライブラリだ。静的ライブラリはコンパイルの際、プログラムの内部に埋め込まれる。共有ライブラリはコンパイルの際、プログラム内部に埋め込まれることはなく、必要なときに、外部にある共有ライブラリを活用しているのだ。

共有ライブラリを使う利点は、プログラムファイルを小さくできることにある。そしてプログラムを動かす際、プログラムファイルをメモリ上に読み込むのだが、プログラムファイルが小さいとメモリの使用量が少なくなるため、メモリの節約になる。そのためコンパイルで静的ライブラリを指定しない限り、共有ライブラリが選択されるのだ。

■ライブラリ依存性の問題

複数のプログラムが、共有部品として共有ライブラリを使っていると、ファイルを軽くしたり、プログラム実行時にメモリの節約といった利点がある。だが問題点もある。



例えば、共有ライブラリが、何らかの障害で消えて無くなったら、それを使っているプログラムが動かなくなる。共有ライブラリのバージョンが上がり関数の仕様が変わった場合でも、様々なソフトが動かなくなるなどの影響が出る。

LinuxでもWindowsでも起こる問題だ。システムにセキュリティーの不具合が発見された際、下手に更新するとライブラリの中身も書き換わり、様々なソフトに影響を及ぼす可能性がある。

Linuxだと暗号化で重要なOpenSSLライブラリが下手に更新できないし、Windowsの場合、出たばかりのHotfixを使うと、不具合が出やすいという話もある。このことは悩ましい問題なのだ。

■メモリ領域の確保

パソコンを触っていたりサーバー管理をしている際、沢山のソフトが稼動しているとメモリ不足になり、動きが重たくなる事がある。プログラムはメモリを使用している。プログラム本体をメモリ領域に読み込むだけでなく、変数などを記憶させるため、メモリ領域を確保しているからだ。

わかりやすい例だと、LinuxやemacsやWindowsのメモ帳といった、文章を書き込むソフトで、人間が入力した内容は一旦、メモリに保管される。入力や書き換えの度に、ファイルに書き出したり、変更すると、処理が遅くなるため、ファイルに保存させる前に、メモリ領域を確保して、メモリ上にデータを置いているのだ。ファイルに書き出すよりも、高速に処理できるからだ。

C言語のプログラムで簡単なメモリ領域の確保を見てみる。

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int *a ;
  a = malloc(10);
  return(0);
}
```

上のプログラムは、10バイトの領域を確保するプログラムだ。mallocはメモリ領域を確保する命令だ。ちなみにmallocは「Memory Allocation」の略で、「メモリの割り当て」という意味なので、そのまんまだ。

世に出ているプログラムは、処理速度を速くするため、処理中のデータをメモリ上に置いている。そのため多くのデータを使って作業している場合、メモリも多くの領域が必要になり、メモリ不足に陥ってしまう問題がある。

色々なソフトを同時に動かしているとパソコンやサーバーが重たくなるのは、メモリ領域の確保のしすぎてメモリ不足に陥っているのが問題なのだ。

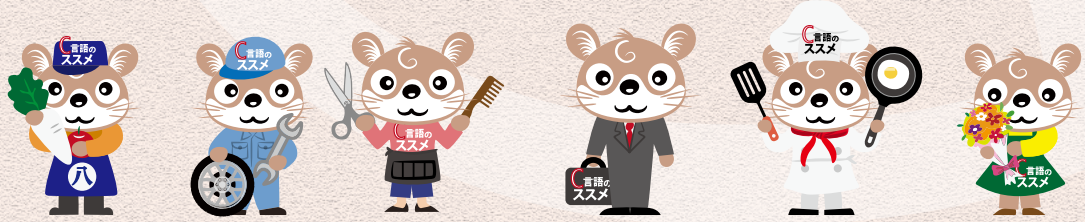
■メモリ違反

パソコンを動かしている際、ソフトが落ちたり、OSが固まったりする事がある。
その原因はメモリ違反だが、簡単なプログラムで再現する事ができる。

```
#include <stdio.h>

int main(void)
{
  int i ;
  int a[4] ;

  for ( i = 0 ; i  < 100 ; i++)
  {
    a[i] = i ;
  }
  return(0);
}
```

a[] という配列に対して、確保した領域をはみ出す量のデータを書き込むプログラムだ。このプログラムを走らせると「セグメンテーション違反です」というエラーが出る。
確保したメモリ領域に対して、それを超える量のデータの書き込みがあると、エラーが出る仕組みなのだ。

普通のプログラムなら、メモリ違反をすると、プログラムの処理が強制終了する。OSに備えられているメモリ保護機能が働くからだ。だが、OS内部のプログラムやハードウェアを制御するドライバが、プログラムの不具合でメモリ違反した場合、メモリ保護機能が働かないためOSの管理で使っているメモリ領域を破壊したりする。このため、正常なデータを失ったOSは、正常に作動できず、OSが固まったりするのだ。



■バッファオーバーフロー攻撃

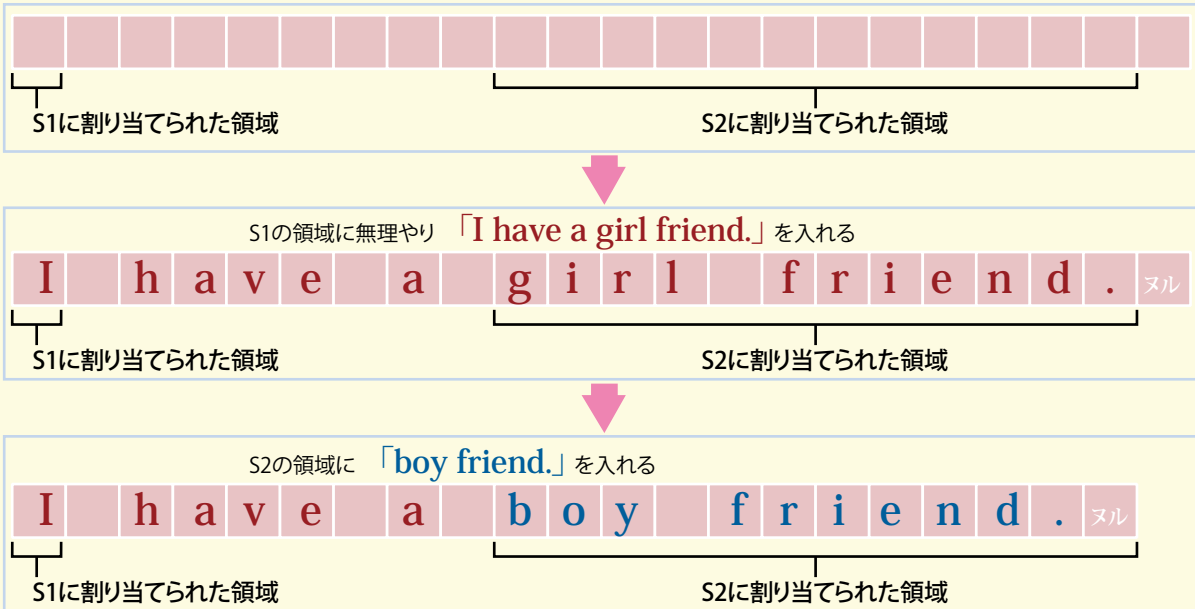
ウイルスを忍ばせたり管理者権限を奪って、システムを乗っ取る手口のひとつだが、C言語のメモリ違反を応用した手口なのだ。C言語のちょっとしたメモリ違反の場合、メモリ保護機能が働かないため、エラーが出ないままプログラムが動き続ける。そのため以下のプログラムが動いてしまう。
(※)あくまでも私が使っているLinuxでの動作結果で、環境によっては結果が異なることもある

```
#include <stdio.h>
#include <string.h>

int main(void)
{
  char s2[12],s1[1] ;

  strcpy(s1,"I have a girl friend.");
  fprintf(stdout,"%s\n",s1);
  strcpy(s2,"boy friend.");
  fprintf(stdout,"%s\n",s1);

  return(0);
}
```

実行結果は以下のようになる

```
[suga@linux]$ sample
I have a girl friend.
I have a boy friend.
[suga@linux]$
```

girlがboyに書き換わる仕組み



s1配列は1文字分のメモリ領域しか確保していない。だが、ちょっとしたメモリ違反だとメモリ保護機能が働かないため、21文字分の文字列を書き込んでもエラーが出ない上、s2配列の領域にまで書き込んでしまっている。その後、s2配列に「boy friend.」の文字を代入すると、s1配列からはみ出した文字列を上書きする形になるため、s1配列に格納されている文字列が「I have a boy friend.」に書き換わるのだ。

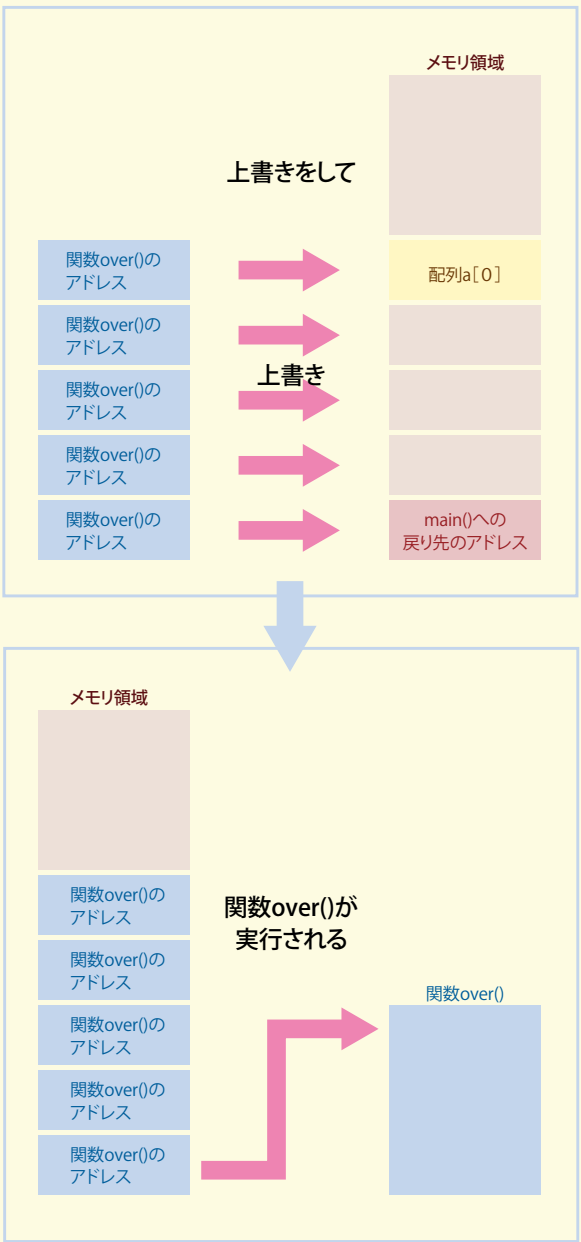
この程度なら笑い話になるが、ちょっとしたメモリ違反は、深刻な問題も引き起こす。それがバッファオーバーフロー攻撃なのだ。まずは説明の前段階として、呼び出す予定のない関数を動かすプログラムを見てみる。

```
#include <stdio.h>

int over(void)
{
  printf("Over Flow!!\n");
  exit(0);
}  /* 呼び出す予定がない関数 */

void func(void)
{
  int a[1];
  int i ;
  for ( i = 0 ; i < 5 ; i++ )
    {
      a[i] = (int)over ;
      /* 配列aに関数over()のアドレスを代入 */
    }
}

int main(void)
{
  func();
  return(0);
}
```

sub()関数は、どこからも呼び出されていないのだが、プログラムを実行すると、over関数が呼び出される。

以下の仕掛けをしているからだ。

メモリ違反による関数の呼び出し



メモリ違反しながら関数over()の開始先を書き込むことで、main()関数への戻り先のアドレスを上書きしてしまう。そのため関数終了時に、main()関数に戻らず、over関数が呼び出されてしまうのだ。これを悪用したのがバッファオーバーフロー攻撃で、ウイルスを忍ばせたり、管理者権限を奪うのだ。

ウイルスをしのばせる手口のイメージ

ウイルスを忍ばせる手口も、管理者権限を奪う手口も、本来なら呼び出し元の関数に戻る場所のアドレスを、変数Aのメモリ上のアドレスに書き換えている。関数終了時の戻り先アドレスが、変数Aのアドレスに書き換わっているため、変数Aのアドレスに飛ぶと、そこに格納されてしまったウイルスプログラムやシェル起動プログラムが呼び出されたりするのだ。

管理者パスワードを強固にしたら大丈夫と思っている人もいるだろう。以前の私もそうだった。だが、管理者のパスワードを盗まなくても、パスワード破りを行わなくても、管理者権限で動くプログラムの欠陥を利用して、バッファオーバーフロー攻撃を行えば、管理者権限が奪えるのだ。

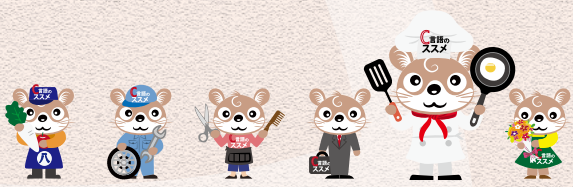
バッファオーバーフロー対策として、LinuxではExec-Shield、Windows7ではデータ実行防止(DEP)の設定で、特定のメモリ上に送り込まれたプログラムを作動させない設定が行える。だがこれによって、動かなくなるソフトもあるようなので、悩ましい問題だ。

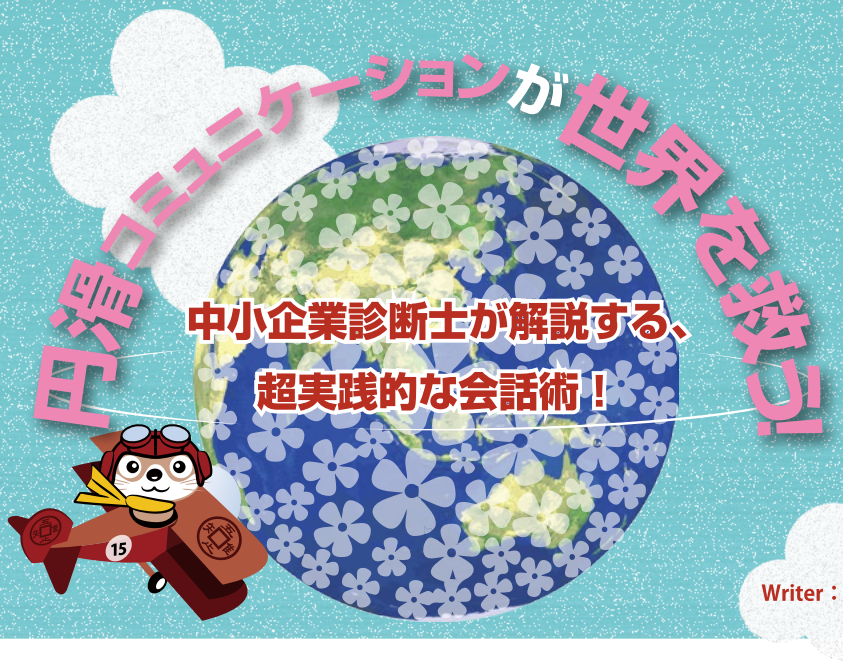
管理者権限を奪う手口のイメージ

C言語を知る事で、システムが抱える問題点が理解しやすい話をした。

それ以外にもC言語を学ぶ利点はある。例として、ソケットを使った簡単な通信プログラムを書いてみると、クライアント・サーバーの概念が具体的に見えてくる。ファイルロックのプログラムを書いてみると、同時書き込み防止の仕組みが見えてくる。

初歩的なC言語のプログラムしか書くことができない私でも、概略を知ることでシステムの勉強になり、この体験から、「C言語でのプログラミングは必要ないから要らない」とは言わず、是非、C言語を触ってみることをお勧めする!





Writer：濱口　誠一

第4回　伝えるためのシンプルすぎる技術「PREP法」

今回は、「PREP」法という伝えるための技術を紹介します。PREP とは、自分の考えを相手にわかりやすく伝えるための技術です。

では、PREP とはどのような内容でしょうか? PREP とは、**P**oint → **R**eason → **E**xample → **P**oint それぞれの頭文字をとったものです。

P ＝要点、結論（Point）
R ＝理由、背景、根拠、効果（Reason）
E ＝具体的な事例、データ（Example）
P ＝結論の再確認（Point）

例えばゴールデンウイークや夏休みなどの大型連休に家族や恋人から「ディズニーランドにでも行こうよ♪」と誘われたときのことを考えてみましょう。本音では「混むから家にいたいよな…」と考えている場合、どう言えば伝わるでしょうか?

P：どこにも行かず家でのんびりすべきだ
R：連休時は人が集中するため、ディズニーランドは異常に込んでおり、逆に疲れるだけ
E：去年行った時にも無茶苦茶込んでいて、結局、１つしかアトラクションを楽しめてない
P：よって、連休中はどこにも行かず家でのんびりしよう

なぜ、ディズニーランドに行くべきでないのか?という理由が明確に伝わりますね!（ただし、これを話した後、命の保障は一切できません…）

では、PREP 法だとなぜ伝わりやすいのでしょうか?

1）結論が最初に来る

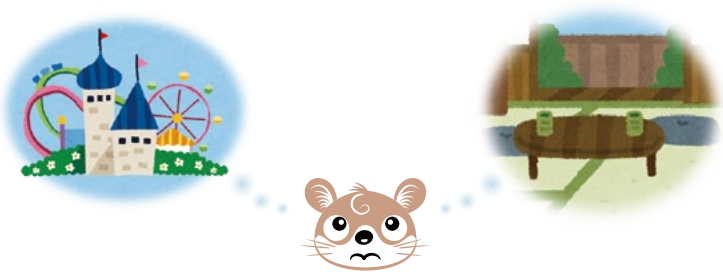
人は知らないものには不安を持ちやすいもの。逆に知っているものであれば安心して話を聞くことができます。最初に P で結論を出すと、聞き手は「何の話をするのか?何を言いたいのか?要求は何か?」がわかるので、安心してその後の話を聞くことができます。

2）理由や思考プロセスがわかる

R で根拠を示すので、なぜその結論が正しいと言えるのか?を理屈のレベルで納得できます。また、話し手がその結論に至った思考プロセスも見えてくるので、話の不透明さがなくなります。

3）具体例によりリアリティを持つことができる

E で実際の事例を出すので、説明内容を頭でイメージしやすくなり、リアリティ（現実感）が醸成されます。

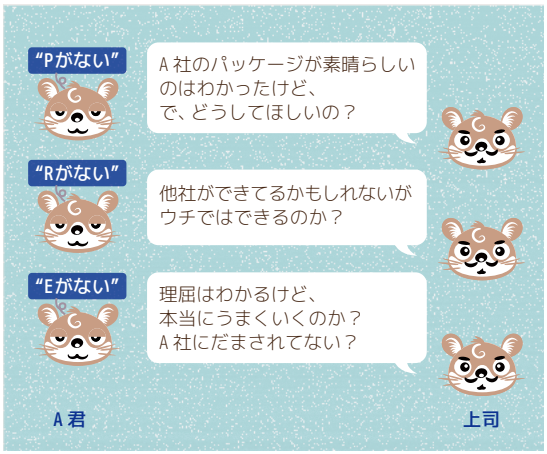


実例で考えてみましょう

営業管理のパッケージシステムを導入する場合を考えてみましょう。

P：A 社の営業管理システムを導入すべきである
R：導入コストが安く、当社の営業スタイルにマッチするので、スムーズに導入できる
E：実際に当社と同規模の B 社、C 社も A 社の営業管理システムを導入している
P：A 社の営業管理システムを導入すべきである

仮に P、R、E がないとどのような状態になるでしょうか?



まとめると、

Point がない　→ イイタイコトが伝わらない
Reason がない　→ 理由・根拠が伝わらない
Example がない　→ 信憑性にかける

という評価をされます。逆に言えば、PREP を意識すると、上に書いたような事態を避けることができます。ちなみに気づかれた方もいらっしゃるかもしれませんが、ここまでの内容は、PREP の形式でご説明しています。結論→理由→具体例、という順番ですね。

自分のクセに気づく

また、PREP を応用して自分のコミュニケーションのクセに気づくこともできます。ここで、ご自身の普段の仕事を振り返ってみてください。上司や顧客からどのようなコメントをされることが多いですか?

よく言われること	PREPの観点による改善策
イイタイコトがよくわからないな。何が言いたいの？どうしてほしいの？	Pointがないので、結論を最初に言うようにする
何でその結論になるの？論理が飛んでない？根拠は何？	Reasonがないので、根拠や自分が結論にいたった思考プロセスを説明する
理屈はわかるけど何かイメージがわかないな／机上の空論にしか聞こえない	Exampleがないので、具体的な事例やデータを説明する

このように、相手のコメントから「自分の伝えるクセ」を知ることができます。自分のコミュニケーションのクセには、なかなか気づきづらいもの。また、コミュニケーションに足らない要素を的確に指摘してくれる人も少ないものです。

そこで、自分の足らない要素を埋めるためにも、よく言われることを PREP の観点で改善されてはいかがでしょうか?

PREP の注意点

ただし、PREP を使う際に 1 つだけ注意点があります。それは、「結論から先に言うことで相手が怒ってしまい、その後に聞く耳を持たない状態にならないかどうか?」です。

今回の最初のディズニーランドの例では…。まあ、言うまでもないですよね（笑）。

私自身は、例えば役員相手に話すときには基本的には PREP で話をします（彼らは結論から先に知りたがる生き物なので）。ただし、役員の意向に沿わない話をする場合には、「ご相談があるのですが、まず状況から説明させてください」といって、あえて E から話すことで、余計な怒りを買わないように気をつけています。

PREP は有効なツールですが、ツールの効果は相手や状況によって変わるもの。「結論を先に言うことで地雷を踏まないか?」というスクリーニングを行ったうえで、ビジネスに是非活用してください。

ISBN 978-4-904807-20-0
C3455 ¥500

9784904807200

USP 研究所
定価（本体 500円+税）

1923455005002