今回は、環境変数について学ぶ。環境変数はシェルが管理する変数である。変数とはいえ C 言語の変数とは全く別の仕組みである。プログラムは実行時に環境変数の値を調べることができる。

1. 環境変数の例

   Mac や UNIX(Linux,FreeBSD) でよく使用される環境変数の「名前」と「値」の例を示す。

```
SHELL=/bin/bash                 # 使用中のシェル
TERM=xterm-256color             # 使用中のターミナルエミュレータ
USER=sigemura                   # 現在のユーザ
PATH=:/usr/bin:/bin:/usr...     # シェルがコマンドを探すディレクトリ一覧
PWD=/Users/sigemura             # カレントディレクトリのパス
HOME=/Users/sigemura            # ユーザのホームディレクトリ
LANG=ja_JP.UTF-8                # ユーザが使用したい言語 (ja_JP.UTF-8(日本語))
LC_TIME=C                       # ユーザが日時の表示に使用したい言語(C 言語標準)
TZ=Japan                        # どの地域の時刻を使用するか (日本)
CLICOLOR=1                      # ls コマンド等がカラー出力する (yes)
```

2. 環境変数を変更した例

   例えば LC_TIME 環境変数は日時の表示に使用する言語を決める。また、TZ 環境変数はどの地域の時刻を表示するかを決める。date コマンド (現在時刻表示)、cal コマンド (カレンダー表示)、ls コマンド (ファイルの最終変更時刻表示) 等がこれらの環境変数の値により日時の表示を変化させる。

```
$ printenv LC_TIME              # 環境変数 LC_TIME の値を確認する
C                               # C 言語標準 (米国英語表記) を使用する
$ date
Sun Jun 26 11:04:27 JST 2016    # 英語表示で日本時間
$ ls -l
-rw-r--r--  1 sigemura  staff    319 Jun 26 10:30 Makefile
$ LC_TIME=ja_JP.UTF-8           # LC_TIME に日本語を表す値をセットして試す
$ date
2016 年 6 月 26 日 日曜日 11 時 03 分 21 秒 JST
$ ls -l
rw-r--r--  1 sigemura  staff    319  6 26 10:30 Makefile
$ export TZ=Cuba                # TZ 環境変数を作ってキューバを示す値をセット
$ date
2016 年 6 月 25 日 土曜日 22 時 03 分 26 秒 JST       # 日本語表示でキューバ時間
$
```

   このようにプログラムの振る舞いを環境変数でコントロールすることができる。

3. 環境変数は誰が決めるか

   (a) システム管理者

   システム管理者はユーザがログインした時の初期状態を決める。UNIX や Mac では、/etc/profile ファイル等に書かれたスクリプトが全ユーザのシェル起動時に実行される。システム管理者は全ユーザに共通の初期化処理をここに書いておく。

   (b) ユーザの設定ファイル

   ユーザは自分のホームディレクトリの.bash_profile ファイルに自分専用の初期化処理を書くことができる。ここに、「環境変数の操作」で説明するコマンドで処理を書いておく。次は.bash_profile ファイルの例である。

```
PATH="/usr/local/bin:$PATH:$HOME/bin:."
export LC_TIME=C
export CLICOLOR=1
```

(c) ユーザによるコマンド操作

シェルのコマンド操作で環境変数を操作することができる。但し、影響範囲は操作したウインドのシェルのみである。次回ログイン時には操作結果の影響は残らない。

4. 環境変数の操作

シェルのコマンド操作で環境変数を操作する。

(a) 環境変数の表示

```
$ printenv              # 全ての環境変数の名前と値を表示する
$ printenv 名前         # 名前の環境変数の値だけを表示する

実行例
$ printenv
SHELL=/bin/bash
TERM=xterm-256color
USER=sigemura
...
$ printenv SHELL
/bin/bash
$
```

(b) 環境変数の操作

i. 新規作成 (その１)

UNIX の標準シェル (sh) では次の２ステップで環境変数を作る。

```
$ 変数名=値             # 一旦､シェル変数として作る
$ export 変数名         # シェル変数を環境変数に変更する

実行例
$ printenv MYNAME       # MYNAME 環境変数を確認 => 存在しない (何も表示されない)
$ MYNAME=sigemura
$ export MYNAME
$ printenv MYNAME       # MYNAME 環境変数を確認 => 値が "sigemura" だとわかる
sigemura
$
```

ii. 新規作成 (その２)

Mac や Linux で使用されるシェル (bash) では、次のように 1 行の操作で環境変数を作ることができる。

```
$ export 変数名=値

実行例
$ printenv MYNAME       # MYNAME 環境変数を確認 => 存在しない (何も表示されない)
$ export MYNAME=sigemura
$ printenv MYNAME       # MYNAME 環境変数を確認 => 値が "sigemura" だとわかる
sigemura
$
```

iii. 値の変更

既に存在する環境変数の値を変更する操作は次の通り。変数名を間違った場合、間違った名前で新しいシェル変数が作成されエラーにならないので注意が必要である。

```
$ 変数名=値

実行例
$ MYNAME=yosinaga          # MYNAME 環境変数の値を変更
$ printenv MYNAME          # MYNAME 環境変数の値が変更されている
yosinaga
$
```

iv. 値の参照

「$変数名」と書くことにより環境変数やシェル変数の値をコマンド入力時に参照できる。例えば、すでにある PATH 環境変数の値に新規ディレクトリを追加する例を示す。

```
$ printenv PATH                        # 初期状態を確認する
/usr/local/bin:/bin:/usr/bin
$ PATH=$PATH:.                         # カレントディレクトリを追加する
$ printenv PATH
/usr/local/bin:/bin:/usr/bin:.
$ PATH=$PATH:$HOME/bin                 # ホームの bin ディレクトリを追加する
$ printenv PATH
/usr/local/bin:/bin:/usr/bin:./User/sigemura/bin
$
```

v. 変数の削除

環境変数、シェル変数のどちらも次の操作で削除できる。存在しない変数を unset してもエラーにならない。変数名を間違ってもエラーにならないので注意が必要である。

```
$ unset 変数名

実行例
$ unset MYNAME
$ printenv MYNAME        # MYNAME 環境変数を確認 => 存在しない (何も表示されない)
$
```

vi. 値を一時的に変更

変数の値を一時的 (今回のコマンド実行の期間だけ) に変更してコマンド (プログラム) を実行する。

```
$ env 変数名=値 ...  コマンド

実行例
$ date
Sun Jul  3 08:35:42 JST 2016
$ env LC_TIME=ja_JP.UTF-8 TZ=Cuba date     # 日本語表示、キューバ時間で date 実行
2016 年 7 月 2 日 土曜日 19 時 36 分 01 秒 CDT
$ date
Sun Jul  3 08:36:05 JST 2016
$
```

5. 環境変数の仕組み

図 1 に環境変数の仕組みを示す。

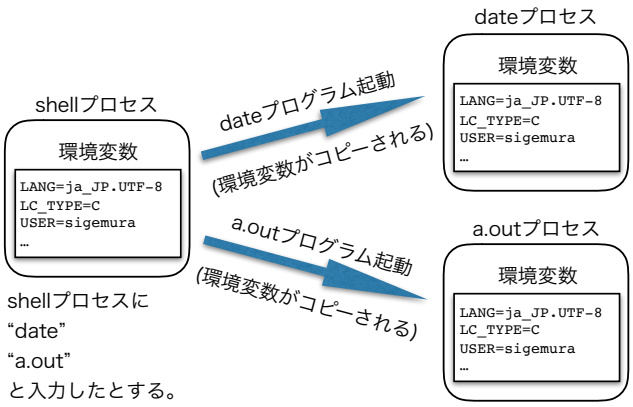

図 1: プログラム起動時の環境変数コピー

シェルプロセス (shell プロセス) は環境変数を自プロセスのメモリ空間で記憶・管理している。上記の「環境変数の操作」で紹介したコマンド等は、シェルのメモリ空間の環境変数・シェル変数を操作するものである。このようにシェル自身が処理するコマンドを**内部コマンド**と呼ぶ。

シェルは入力されたコマンドが内部コマンド以外 (**外部コマンド**) なら、コマンド名と同じ名前のプログラムを探し子プロセスとして起動する。この時シェルは、自身の環境変数を子プロセスにコピーするように OS カーネルに依頼する。OS カーネルは子プロセスのメモリ空間のどこか（例えばスタックの底）に環境変数をコピーする。子プロセスはメモリ空間内の環境変数のコピーを参照・変更・削除できる。

この仕組みはシェルに限らず全ての**他のプログラムを起動するプログラム**で用いられる仕組みである。実は上記の「環境変数の操作」で紹介した操作方法で `env` コマンドだけは**外部コマンド**である。`env` コマンドは**他のプログラムを起動するプログラム**として実装できる。

図 2 に `env` コマンドの仕組みを示す。`env` コマンドは自身の環境変数を変更した後、目的のコマンドを起動する。

6. C 言語プログラムから環境変数を参照する方法

C 言語プログラムから環境変数を参照するために、以下に紹介する二つの方法が使用できる。

(a) main 関数の仮引数やグローバル変数 environ を使用する方法

C 言語の `main` 関数には実は第 3 引数が存在した。`environ` という名前の C 言語のグローバル変数が存在する。

```
extern char **environ;  // どこかで定義されている environ 変数を使用する準備
int main(int argc, char *argv[], char *envp[]) { ...
```

図 3 にプロセスのメモリ空間にコピーされた環境変数のデータ構造を示す。

例えば全ての環境変数を表示する C プログラムは次のように書くことができる。

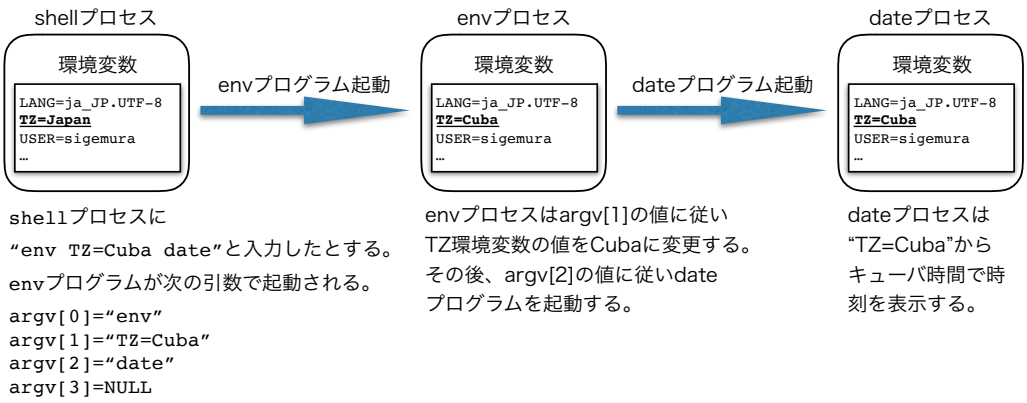

図 2: env プログラムの仕組み

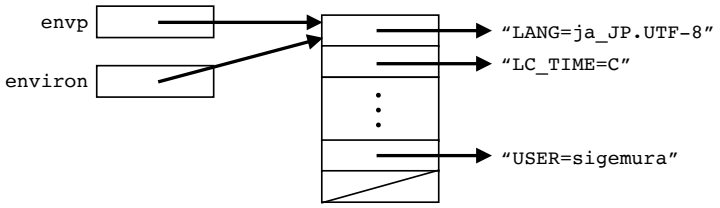

図 3: 環境変数のデータ構造

```
// 全ての環境変数を印刷するプログラム (envtest1.c)
#include <stdio.h>
extern char **environ;

int main(int argc, char*argv[], char*envp[]) {
  for (int i=1; environ[i]!=NULL; i++)
    printf("%s\n", environ[i]);
  return 0;
}

/* 実行例
$ envtest1
TERM=xterm-256color
SHELL=/bin/bash
CLICOLOR=1
...
*/
```

(b) getenv 関数を使用する方法

C 言語標準ライブラリの getenv 関数は次のような書式の関数である。

```
書式
#include <stdlib.h>
char *getenv(char *name);
```

name には環境変数名を渡す。getenv は環境変数の値を示す文字列を指すポインタを返す。name の環境変数が見つからない場合は NULL ポインタを返す。次の C プログラム

は LANG 環境変数の値を表示するものである。

```
// LANG 環境変数の値を表示するプログラム (envtest2.c)
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char*argv[], char*envp[]) {
  char *val;
  val = getenv("LANG");
  if (val!=NULL)
    printf("LANG=%s\n", val);
  else
    printf("LANG␣does␣not␣exist.\n");
  return 0;
}

/* 実行例
$ envtest2
LANG=ja_JP.UTF-8
 */
```

7. C 言語プログラムから環境変数を操作する方法

自プロセスのメモリ空間に置かれた環境変数のコピーを操作する三つの C 言語標準ライブラリ関数を紹介する。

(a) setenv 関数

環境変数を新規に作成したり、値を上書きしたりする関数である。overwrite が 0 の時は新規作成専用になる。

```
書式
#include <stdlib.h>
int setenv(char *name, char *val, int overwrite);

使用例
setenv("MYNAME", "sigemura", 1);
```

name は環境変数の名前、val は環境変数にセットする値である。overwrite は、0 の時に上書き禁止、それ以外の時に上書き許可を意味する。setenv の返す値が 0 は正常、それ以外はエラーである。

(b) putenv 関数

環境変数を新規に作成したり、値を上書きしたりする関数である。

```
書式
#include <stdlib.h>
int putenv(char *string);

使用例
putenv("MYNAME=sigemura");
```

string は NAME=VALUE 形式の文字列である (それ以外の形式の文字列を渡すとエラーになる)。putenv の返す値は 0 が正常、それ以外はエラーである。putenv("NAME=VALUE"); は、setenv("NAME","VALUE",1); と同じ操作を行う。

(c) unsetenv 関数

環境変数を削除する関数である。

書式
```
#include <stdlib.h>
int unsetenv(char *name);
```

使用例
```
unsetenv("MYNAME");
```

name は環境変数の値である。unsetenv の返す値は 0 が正常、それ以外はエラーである。

8. 宿題

(a) 上記の実行例を試してみる

環境変数を操作するシェルの**内部コマンド**、**外部コマンド**を使用してみる。下の囲み記事を参考に、LC_TIME 環境変数や TZ 環境変数を色々試すと面白い。例えば、「モスクワ時間、ロシア語表示」で時刻を表示するにはどうしたらよいか？

(b) myprintenv プログラム

printenv と同様な働きをする myprintenv プログラムを作成する。作成したプログラムを印刷して提出する。

〆切：7 月 8 日 (金)

ロケール

LANG 環境変数や LC_TIME 環境変数にセットする値をロケール名と呼ぶ。ロケール名は "言語コード_国名コード. エンコーディング"の組み合わせで表現される。
言語コードは ISO639 で定義された 2 文字コードである (日本語は"ja")。詳しくは https://ja.wikipedia.org/wiki/ISO_639-1 等を参照のこと。
国名コードは ISO3166 で定義された 2 文字コードである (日本は"JP")。詳しくは https://ja.wikipedia.org/wiki/ISO_3166-1 等を参照のこと。
エンコーディングは、使用する文字コードの符号化方式を示す。エンコーディング方式がターミナルエミュレータのテキストエンコーディングと一致していないと文字化けを起こす。使用可能なロケールの一覧は locale -a コマンドで表示できる。

タイムゾーン

TZ 環境変数にタイムゾーンを表す値をセットする。MacOS や UNIX の内部では時刻は協定世界時 (UTC) で管理されている。時刻を表示する時に現地時間に変換して表示する。
時刻に関係するプログラムは協定世界時と現地時間の変換方法を TZ 環境変数から知ることができる。日本の場合はタイムゾーン名が JST、協定世界時との時差が −9 時間なので、TZ=JST-9 となる。
この形式の他に/usr/share/zoneinfo/に置いてあるファイル名でタイムゾーンを指定することもできる。/usr/share/zoneinfo/Cuba ファイルが存在するので TZ=Cuba と指定できる。/usr/share/zoneinfo/Asia/Tokyo ファイルが存在するので TZ=Asia/Tokyo と指定できる。