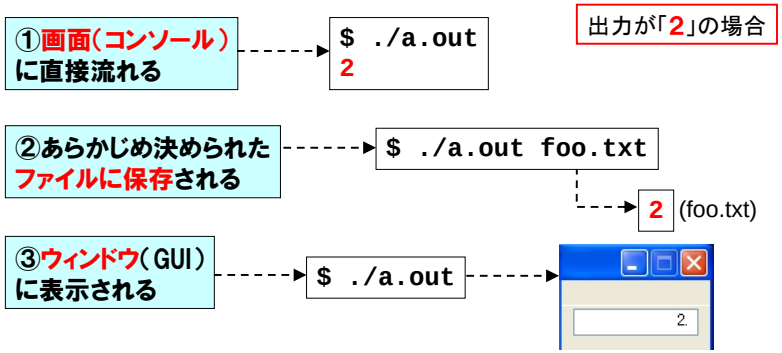
プログラミング言語II
①フィルタの概念とシェルの基礎

阿萬裕久
aman@cs.ehime-u.ac.jp

プログラムからの出力

- 通常，プログラムを実行すると，その**結果が何らかのかたちで出力**される：主なパターンは三つ



①は②を兼ねることが可能

- **リダイレクト**を使う

① `$ ./a.out` / `2` → `$ ./a.out > foo.txt`

- こうすると，出力は画面（コンソール）に流れず，その**代わりに foo.txt というファイルへ流れる**
- もちろん出力先のファイルは自由に指定できる
  そのため**無理に②形式のプログラムを作る必要はない**

（参考）上書きではなく追加したい場合

- リダイレクトを行う場合
  - **＞ は上書き**となるが

```
$ ./a.out > foo.txt
$ ./a.out > foo.txt
```

(foo.txt) 2

1回目で「2」が入るが，2回目で上書きされる

  - **＞＞ とすれば追加**となる

```
$ ./a.out >> foo.txt
$ ./a.out >> foo.txt
```

(foo.txt) 2 2

1回目で「2」が入り，2回目で追加される

※実行前には foo.txt が存在しない場合

①タイプと③タイプの比較

	①コンソール出力	③ウィンドウ出力
長所	**実行結果を他のプログラムの入力にそのまま使える**	実行結果が，ユーザにとって分かりやすい
短所	表示が地味である 誤ってデータを上書きする可能性がある	実行結果の記録が面倒（手で書き写すことになる） 実行結果を他のプログラムに引き渡せない

ここでは，「実行結果を他のプログラムの入力に使う」という特長に注目する

【例題１】

いま，データファイル ex1.txt の中に整数がいくつか記録されている（一行に一つずつ）．
この中からの最大値を見つけ出して出力しなさい．

- C言語でプログラムを書いてもよいが，ここでは **Unix のコマンドだけでこれを実現**してみる
- 使用するコマンド
  - sort
  - tail

【例題１】（答え）

```
sort -n ex1.txt | tail -n 1
```

- sort コマンド
  与えられたファイルの**内容をソーティングして出力**する
  通常は辞書（電話帳）式だが，-n オプションで数値順に
- tail コマンド
  与えられたファイルの**末尾を出力**する
  -n オプションで行数を指定（-n 1 ならば最後の一行）

解説（１／５）

- まず，sort コマンドの働き

```
sort -n ex1.txt
```

```
1
5
7
8
10
12
....
```

ex1.txt の中に書いてある数字を小さい順に出力してくれる

- sort コマンドは ex1.txt の内容をソーティングし，画面（コンソール）へそのまま出力する
- 通常は文字列のソーティング（並べ替え）を行うが，-n オプションを付けて実行すれば数値として扱ってくれる

解説(2／5)

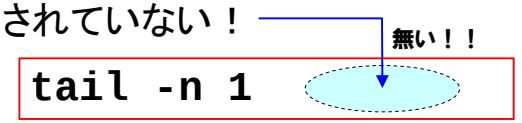
- 次に，tail コマンドの働き

```
tail -n 1
```

- tail コマンドは，与えられたファイルの末尾のみを画面(コンソール)へ出力する
- 後ろからいくつの行を取り出すかは -n オプションで指定する

解説(3／5)

- ここでちょっと疑問かも？
- tail コマンドには入力ファイル(ex1.txt とか)が指定されていない！

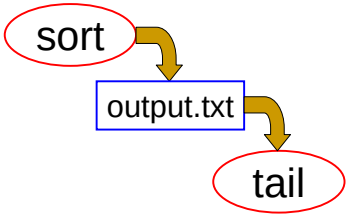


- その場合は自動的に**キーボードからの入力**を受け付けるようになっている
  - Cプログラムでの scanf の動きと同じ
  - 入力の終了は [Ctrl] + [d]

解説(4／5)

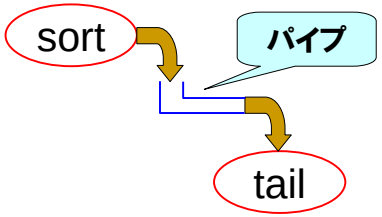
- いったんファイルを経由すれば，手作業(キーボード)での入力の手間を省ける

```
sort -n ex1.txt > output.txt
tail -n 1 < output.txt
```



解説(5／5)

- ファイルを使わず一気に流し込んでしまおう



“ | ”を使えば，**左側の出力を一気に右側の入力へつなぐ**ことができる
(これを「**パイプ**」という)

```
sort -n ex1.txt | tail -n 1
```

フィルタ(filter)

- 入力はキーボード(正確には**標準入力**)から行う
  ※C でいえば scanf を使用
- 出力は実行画面(正確には**標準出力**)へ行う
  ※C でいえば printf を使用

- このタイプのプログラムを**フィルタ**という
  Unix コマンドにはフィルタが多く存在する

プログラムをフィルタにする利点

- プログラムどうしを組合わせて使いやすい
  (例)ex1.txt の中で三番目に大きな値を出力

```
sort -n ex1.txt | tail -n 3 | head -n 1
```

sort でデータを並べ替えて出力 → tail で最後の三つだけを出力 → head で最初の一つだけを出力

並べ替えた後，最後の三つだけに絞り込み，さらにその中の最初の一つを出力しているので，全体の中で三番目に大きな値が得られる

データ処理でよく使うコマンド

- cat
- head
- tail
- wc
- grep
- cut
- tr
- sort
- uniq
- join
- fold
- find

シェルスクリプト

- 通常，プログラムの実行は，画面(コンソール)上で**コマンドやプログラム名を入力**する
  - コマンドやプログラムを受け付ける部分を**シェル**(shell)という
  - シェルは入力を解釈し，その裏方にいる**OS**(**カーネル**)へ処理を依頼する
- 普段，キーボードで入力している**コマンド列をファイルに書き出したもの**を**シェルスクリプト**(shell script)という

シェルスクリプトの例

- sample.sh

```
whoami
date
cal
```

- 実行

```
bash sample.sh
```

- 要は実行したいコマンドを**テキストファイルに書いて**
- **bash コマンドで実行**

※シェルにはいくつか種類があり，
皆さんがいま使っているシェルは，
bash（バッシュ）というものです

※その他の種類
csh（シーシェル），tcsh（ティーシーシェル），
zsh（ゼットシェル）など

【例題２】

ファイル（ただし，テキストファイルに限る）が与えられると，そのファイル名，先頭と末尾の内容（三行ずつ）を表示するスクリプトを作りなさい．

- 実行例

```
bash digest.sh foo.c
```

```
=== foo.c ===
#include

int main(void)
.....

   return 0;
}
```

（先頭の3行）（末尾の3行）

【例題２】（答え）

```
echo "=== $1 ==="
head -n 3 $1
echo
echo "....."
echo
tail -n 3 $1
```

1番目のコマンドライン引数を意味する

- echo コマンド
  指定された文字列を表示する

解説（１／４）

- まず，コマンドライン引数で与えられたファイル名を表示する

```
echo "=== $1 ==="
head -n 3 $1
echo
echo "....."
echo
tail -n 3 $1
```

解説(2／4)

- 次に，そのファイルの先頭部分(3行分)を head コマンドを使って表示する

```
echo "=== $1 ==="
head -n 3 $1
echo
echo "....."
echo
tail -n 3 $1
```


解説(3／4)

- 途中を「省略」していることを示すために「….」を表示するが，その前後も一行ずつ空ける

```
echo "=== $1 ==="
head -n 3 $1
echo
echo "....."
echo
tail -n 3 $1
```


解説(4／4)

- 最後にファイルの末尾部分(3行分)を tail コマンドを使って表示する

```
echo "=== $1 ==="
head -n 3 $1
echo
echo "....."
echo
tail -n 3 $1
```


【例題3】

コマンドプロンプトをどのようなかたちで表示するのかはシェル変数 PS1 に設定されている．プロンプトにユーザ名も表示するように PS1 を設定しなさい．その際，プロンプトは赤色で表示しなさい．

- 表示例

  e0701aman$

シェルの中ではプログラムと同様に変数を使うことができる．
変数の設定は
　　変数名=”値”
で行うことになる

【例題3】(答え)

```
PS1="¥[¥033[31m¥u¥033[0m¥]$ "
```

- 特殊記号 ¥[… ¥]
  修飾部分をこれで囲む
- 特殊記号 ¥033[31m
  これ以降を赤色にする(¥033[0m でリセット)
- 特殊記号 ¥u
  現在のユーザ名を表示する

解説(1/3)

- まず,基本としては PS1 に代入した内容がそのままプロンプトに使われる

```
PS1="Hello$ "
```

→ Hello$

- ユーザ名を自動的に入れる場合は,定義済みの特殊記号である ¥u を使う

```
PS1="¥u$ "
```

→ e0701aman$

解説(2/3)

- ひとまず,これで一部は理解できるであろう

```
PS1="¥[¥033[31m¥u¥033[0m¥]$ "
```

- 次に,¥u の部分に色を付けていく
- まず,そのような修飾を行う場合は ¥[と ¥] で囲むことになっている

```
PS1="¥[¥033[31m¥u¥033[0m¥]$ "
```


解説(3/3)

- 続いて,¥033[31m と書くと,これ以降を赤色に塗るという意味になる

```
PS1="¥[¥033[31m¥u¥033[0m¥]$ "
```

- これで止めると,以降がすべて赤になってしまうので ¥033[0m と書いて,残りをリセットする

```
PS1="¥[¥033[31m¥u¥033[0m¥]$ "
```

参考までに

- ¥033[0m でリセットするのを忘れると，後に続く部分（キーボードからの入力も）すべて赤色になってしまう.
- 31 が赤であることは説明したが，色に関しては 31～37, 41～47 が使える.
- 自分で設定したプロンプトを常に使いたい場合は，~/.bashrc の末尾に PS1="...." を書いておくとよい.

まとめ

- 標準入力と標準出力を入出力としたプログラムを**フィルタ**という
- フィルタを活用することで**プログラム部品の組合わせ**がやりやすい
  - 特に Unix コマンドの組合わせは応用が多い
- 複雑な組合わせや一連の処理は，**シェルスクリプト**にまとめるとよい
- **シェルにも変数**があり，その挙動に影響を与えるものもある（例：プロンプト変数 PS1）