C-プログラミングコース
ポインタ自由自在（1）

Yasuhiro Noguchi

the Research Institute for Embedded Systems Architects (RIESA)

Faculty of Informatics, Shizuoka University

HEPT Consortium

意図・目標

- 意図・目標
  - ポインタの概念を学習し，基本型のポインタ及び配列を扱う関数を設計・実装できるようになります（ただし，ポインタ自由自在（1）では動的メモリ確保は扱わない）.
  - Genericポインタ(void*)を学習して，任意の型に対応するように一般化した関数を設計・実装します.
- 学習項目
  - 変数とポインタ，ポインタ演算子
  - sizeof演算子とポインタ
  - 関数とポインタ（関数の引数，返り値として）
  - ポインタと配列
  - Genericポインタ(void*)

本日の到達目標課題

問題：配列を使ったスタック（push, pop, peek, isEmpty）を設計・実装します．ライブラリとして使いたいので，“複数のスタックを使いたい”，という要望に応えてください．

1. int型を対象としたスタック関数及びそのテストを設計してください．設計した関数を実装しテストしてください．
2. 文字列（要するにchar型のポインタ）を対象としたスタック関数及びそのテストを設計してください．設計した関数を実装しテストしてください．
3. 任意の型に対応したスタック関数及びそのテストを設計してください．設計した関数を実装しテストしてください．

開発環境

- Eclipse IDE for C/C++ Developers
  - http://www.eclipse.org/
- Cygwin
  - http://www.cygwin.org/

変数とポインタ

意図・目標

- 基本型変数とポインタ変数のモデルをイメージできる.
- 変数（基本型変数とポインタ変数の両方）と演算子を使った演算操作をイメージできる.
  - 基本型の変数と，ポインタとの関係を把握することで，任意の変数とポインタのモデルをイメージできるようになる.
  - ポインタ演算子の役割を把握し，ソースコード上の式とモデルとの対応付けができるようになる.
  - ポインタを含む変数とメモリ領域の関係を把握することで sizeof 演算子を扱えるようになる

ポインタの演算

- ポインタに整数nの加算を行うとデータ型のByte数 * n だけポインタのアドレスが変化する（＝型の違いは自動的に加味されます）

```
short arr[] = {1,2,3};
short* arr_ptr = &arr[0];
printf( “%d¥n”, *arr_ptr );
printf( “%d¥n”, *(arr_ptr+1));
printf( “%d¥n”, *(arr_ptr+2));
```

arr[0]
arr[1]
arr[2]
arr_ptr

address	value
0x1000	0xFF
0x1001	0x00
0x1002	0x21
0x1003	0xF3
0x1004	0xE0
0x1005	0xA3
0x1006	0x00
0x1007	0x00
0x1008	0x10
0x1009	0x00
0x100a	0x19
0x100b	0xA7
0x100c	0xff
0x100d	0xcf

サンプルですので詳細は省略します

関数とポインタ

意図・目標

- 引数にポインタを用いることで，呼び出し元に副作用を与えることのできる関数を設計します．
- 戻り値にポインタを用いた場合に，呼び出し元に安心して利用できる戻り値と，典型的なバグの原因となる戻り値の違いを把握します．
- 典型的な引数・戻り値のパターンの利点と欠点を整理します．

サンプルですので詳細は省略します

配列とポインタ

意図・目標

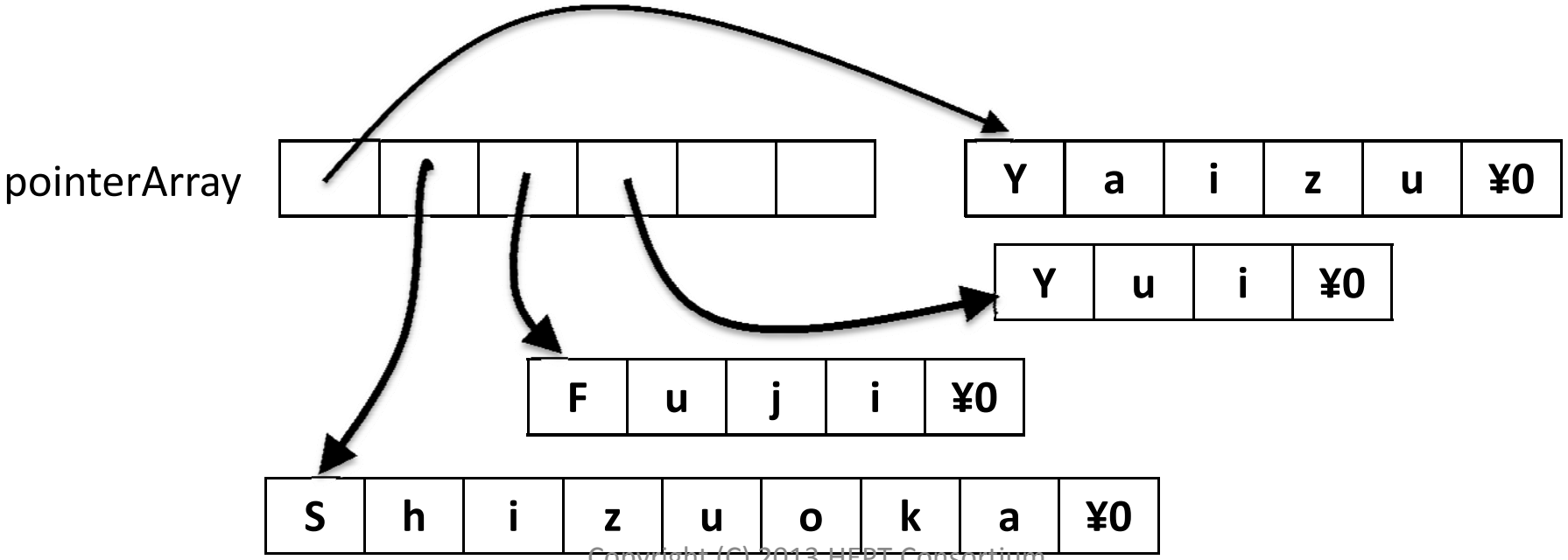
- 配列の添字のインクリメントとポインタのインクリメントが等価であることを学習します.
- 多次元配列（特に2次元配列）とポインタの配列について学習し，その違いを把握することで，必要に応じて適したデータ構造を選択し，また，適切な方法で操作できるようになる.
- 関数の引数として配列を指定した場合の典型的なインタフェース，配列操作のパターンを学習し，適切に関数を設計できるようになる.

サンプルですので詳細は省略します

多次元配列とポインタの配列

```
char** pp = pointerArray;
printf("%s¥n", *pp);
printf("%s¥n", *(pp+1));
printf(“%s¥n”, *(pp+2));
```

2次元配列とは異なり
ポインタの配列だと
この操作ができる



サンプルですので詳細は省略します

GENERICポインタ(VOID*)

意図・目標

- Generic ポインタ (void*) と 基本型のポインタとの違いを理解し，Genericポインタを使ったプログラミングで気をつけなければならない点を整理します．
- 異なる引数型の関数を（もし，それが同等の機能を実現するもので，異なる関数になっているのが型の差異によるものであるならば）Generic ポインタ (void*)を活用して任意の型に対応するように一般化した関数へと再設計します．

サンプルですので詳細は省略します

まとめ

- ポインタ自由自在（1）の意図・目標
  - ポインタの概念を学習し，基本型のポインタ及び配列を扱う関数を設計・実装できるようになります（ただし，ポインタ自由自在（1）では動的メモリ確保は扱わない）.
  - Genericポインタ(void*)を学習して，任意の型に対応するように一般化した関数を設計・実装します.
- 今後の予定（ポインタに関わるところでは）
  - 関数ポインタ　→　関数自由自在
  - 動的メモリ確保　→　ポインタ自由自在（2）

参考文献

- B.W. カーニハン (著), D.M. リッチー (著), 石田晴久 (訳) : “プログラミング言語C”, 共立出版, 1989.
- Steve Oualline (著), 望月康司 (訳), 谷口功 (訳) : “C実践プログラミング”, オライリー・ジャパン, 1998.
- Peter van der Linden (著), 梅原系 (訳) : “エキスパートCプログラミング—知られざるCの深層”, アスキー, 1996.
- 結城浩: “新版C言語プログラミングレッスン入門編”, ソフトバンククリエイティブ, 2006.
- 結城浩: “新版C言語プログラミングレッスン文法編”, ソフトバンククリエイティブ, 2006.
- ニクラウス・ヴィルト, 片山卓也: “アルゴリズム+データ構造 = プログラム”,日本コンピュータ協会, 2002.