プログラム理論と言語　第7回
—ポインタと配列,高階関数,まとめ—

有村博紀
吉岡真治

公開スライドPDF（情報知識ネットワーク研HP/授業）
http://www-ikn.ist.hokudai.ac.jp/~arim/pub/proriron/

本スライドは，2015北海道大学　吉岡真治「プログラム理論と言語,」に基づいて，現著者の承諾のもとに，改訂者（有村）
が加筆修正しています．本スライドに誤り等があった場合には，内容の文責は改訂者にありますのでご注意ください．
2016北海道大学，有村博紀

- 配列へのポインタ
  - char x[4]; shor int y[3];
  - px = &x[0]; py=&y[0]
  - *(px +1)はx[1]と同じ、*(py+1)はy[1]と同じ
  - ポインタのタイプに応じて、アドレスに加えられる値が異なる

多次元配列

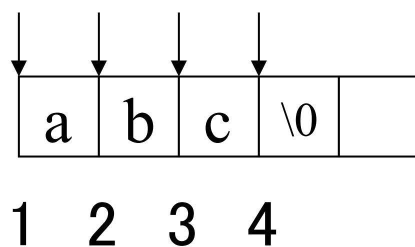
- 多次元配列：配列を配列としたもの
  - 一次元配列の特殊系としてアクセス

```
#include<stdio.h>
void main()
{
  int i,j;
  int a[2][12] = {{31,28,31,30, 31,30,31,31,30,31,30,31},
                  {31,29,31,30, 31,30,31,31,30,31,30,31}};
  for(i =0; i < 2; i++) {
    for(j =0; j < 12; j++) {
      printf("%d %d %d\n", i, j, a[i][j]);
      printf("%d %d %d\n", i, j, *(a[i] + j));
      printf("%d %d %d\n", i, j, *(((int*)(a + i)) + j));     }
  }
}
```

文字列とポインタ

- 文字列はchar型の配列として取り扱う
  - 文字列の最後には、文字列の最後を示すヌル文字¥0が登録される。
  - そのため、配列の長さは、文字数＋1となる。
  - char* a = “abc”
  - char a[] = “abc”

char



文字列のポインタ

- 格納する文字の数によって、必要なメモリ領域が異なるので、doubleやintなどのように固定長の領域をとることができない。
- あらかじめ、必要な領域を確保して、文字列を記述し、文字列の終わりを示す記号"\0"で終端を表す。

```
int strlen(char *a){
  int count =0;
  while(*a != '\0') {
    a++;
    count++;
  }
  return count;
}
```

15400

文字列のソート

- intやfloat型のソート、データを格納している領域の入れ替えが単純な代入操作で可能
- 文字列は、データを格納する領域が大きく、単純にコピーをしてデータを並べ替えるのはコストがかかる。
- ポインタ変数を用いた並べ替え
  - データを格納する領域を並べ替えるのではなく、データの格納されている場所（ポインタ）を並び替える。

2つの並べ替え方

- 長さ10のchar型の配列を4個用意
  - char data[4][10]
  - data = 15400 (char型のポインタ)

Address										
15400	a	b	c	\0						
15410	c	c	b	\0						
15420	a	r	b	\0						
15430	a	a	a	a	a	\0				
15440										

strcpyによる並べ替え

Address										
15400	a	a	a	a	a	\0				
15410	a	b	c	\0						
15420	a	r	b	\0						
15430	c	c	b	\0						
15440										

ポインタの並べ替え

15430
15400
15420
15410

- ポインタを保存するための配列を作成

```
char data[4][10]
char *dataPtr[4];
int i;
for(i = 0; i < 4; i++){
  dataPtr[i] = data[i];
}
```

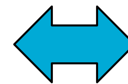
- dataPtrをdataを参照しながら並べ替える
- dataPtrの順番に従って出力

多次元配列では、最大次元以外の要素は
計算で求まるため、上記の場合に、
data[2]=data[3]
といった操作はできない。

Whileとif文

- if文
  - 複数の命令を行う場合に{}を利用

```
if(...)
   operation1;
operation2;
```



```
if(...){
   operation1;
   operation2;
 }
```

- while文
  - 条件を満たす間、operation1を実行し、条件を満たさない時に、ループを抜ける。

```
while(条件){
   operation1;
 }
```

- 基本的なデータ構造の組み合わせ
  - 基本データ:文字列、数値
  - 組み合わせのデータ
    - 学生:名前（文字列）＋学生証番号（数値）
    - 語と出現回数:語（文字列）＋出現回数（数値）

Cにおけるデータ構造の定義

- 構造体
  - struct宣言子による定義
  - 構造体のメンバーを列挙
    - 名前と変数のタイプを指定
    - 例 : 文字の出現回数

```
struct wordCount
{
  char *word;
  int count;
};
```

- 構造体の初期化
- 構造体のメンバーの参照
  - .演算子(ドット演算子)
  - ->演算子(アロー演算子)

```
int main(int argc, char *argv[]){
  struct wordCount wc = { "abc", 1};
  char word[4] ="def";
  char word2[4] ="ghi";
  printf("word: %s count: %d\n", wc.word, wc.count);
  wc.word = word;
  wc.count = 5;
  printf("word: %s count: %d\n", wc.word, wc.count);
  (&wc)->word = word2;
  (&wc)->count = 3;
  printf("word: %s count: %d\n", wc.word, wc.count);
  return 0;
}
```

構造体を利用したプログラム

- ソート済みの文字列の出現回数でソート
  - qsortライブラリを利用
    - 関数へのポインタ(高階関数が利用可能)
  - 出現回数の大小で並べ替え、同数の場合は文字列の順序で並べ替え

```
int wordComp(const void *a, const void *b)
{
  if (((struct wordCount*) a)->count == ((struct wordCount*) b)->count)
    return strcmp(((struct wordCount*) a)->word, ((struct wordCount*) b)->word);
  else
    return ((struct wordCount*) a)->count - ((struct wordCount*) b)->count;
}

メイン関数からの呼び出し
  struct wordCount wc[MAXSIZE];
  qsort(wc, i, sizeof(struct wordCount), *wordComp);
```

高階関数

- 関数を引数とする関数
  - 一般に、関数を引数としない関数を一階関数、関数を引数としない関数のみを関数の引数とする関数を二階関数と呼ぶ
  - 高階関数とは、関数を引数とする関数を繰り返し参照しながら定義できる関数

高階関数の例

- 全ての要素に対して、一定の操作を行う関数
- iterate(f,c,x) = if is_nil(x) then c
  else f(first(x), iterate(f,c,rest(x))
- 実行過程
  - sum(x, y) = x + y, count(x, y) = 1 + y, times(x,y)=x*y
  - iterate(sum, 0, (1 2))= sum(1, iterate(sum,0, (2)))
    = sum(1, 2 + iterate(sum 0 nil))
    = sum(1, 2 + 0)
    = 3
  - iterate(count, 0, (1 2))= count (1, iterate(count,0, (2)))
    = count (1, 1 + iterate(count 0 nil))
    = count (1, 1 + 0)
    = 2
  - iterate(times, 1, (1 2))= count (1, iterate(times,1, (2)))
    = count (1, 2 * iterate(times 1 nil))
    = count (1, 2*1)
    = 2

高階関数の例

- 全ての要素に対して、成立するかどうかを調べる
- forall(f,x) = if is_nil(x) then true
  else if f(first(x)) then forall(f,c,rest(x))
  else false
- 実行過程
  - even(x) = if mod(x, 2) = 0 then true else false
  - forall(even, (2 4 5 6)) = forall(even, (4 5 6))
    = forall(even, (5 6))
    = false
  - forall(even, (2 4 6)) = forall(even, (4 6))
    = forall(even, (6))
    = forall(even, nil)
    = true

C言語による高階関数

- 関数ポインタ
  - 関数の名前と引数を組み合わせて実行する。
  - qsort関数は、関数ポインタを利用して、高階関数のような処理を行っている。

比較:Cのクイックソート(関数へのポインタ)

QSORT(3) Linux Programmer's Manual QSORT(3)

名前 qsort - 配列を並べ変える

書式

```
#include <stdlib.h>
void qsort(void *base, size_t nmemb, size_t size,
  int(*compar)(const void *, const void *));
```

説明

qsort() 関数は、大きさ size の nmemb 個の要素をもつ配列を並べ変える。base 引数は配列の先頭へのポインタである。

compar をポインタとする比較関数によって、配列の中身は昇順 (値の大きいものほど後に並ぶ順番) に並べられる。比較関数の引数は比較されるふたつのオブジェクトのポインタである。

比較関数は、第一引数が第二引数に対して、1) 小さい、2) 等しい、3) 大きい のそれぞれに応じて、1) ゼロより小さい整数、2) ゼロ、3) ゼロより大きい整数のいずれかを返さなければならない。ふたつの要素の比較結果が等しいとき、並べ変えた後の配列では、ふたつの順序は定義されていない。

返り値 **qsort()** は値を返さない。

注意 compar 引数に使用するのに適しているライブラリルーチンとして、strcmp, alphasort, versionsort がある。

- 様々なプログラミング言語
  - 手続き型言語、関数型言語、オブジェクト指向言語…
- 同じ作業が様々な形式で表現可能
  - 人間に理解しやすい言語
  - 計算機にとって効率の良い言語

手続き型言語と関数型言語

- 共通点
  - 手続きの塊を抽象化
    - 関数の定義と再利用
- 異なる点
  - 手続き型言語
    - 逐次的に行う作業を手順として記述し実行
  - 関数型言語
    - 再帰などによる関数定義を用い、関数の解釈（簡約）により作業が実行

言語の考え方と個別言語の仕様

- 実際のプログラミング言語はいろいろな考え方が応用可能
  - 手続き型言語でも再帰プログラミングは可能
  - Lispは、変数の定義や逐次作業の実行といった手続き型言語の考え方が入っている

- 後半部分と合わせて、学期の最後に試験をします。
- 語句説明と授業の最後にやった課題でやったような内容が出題されます。