第１回　基本的データ構造

【この講義について】

1. 担当教員

富井 尚志 (tommy@ynu.ac.jp)

※この資料は森辰則先生と長尾智晴先生が作成されたものを改訂したものです。

2. シラバス

（１）授業のねらい・目的

プログラムとは、データの表現と構造に基づいて、抽象的・数学的な基本解法（アルゴリズム）を具体的に形式化したものである。したがって、計算機を動作させるためには、そのための基本となるアルゴリズムの理解が必須である。本講義では、アルゴリズムの考え方、記述方式、そのために必要な抽象的データ構造、アルゴリズムの数学的解析手法と評価について学び、計算と問題解決の原理について習得する。

（２）授業内容・方法（毎回の講義内容は後述．これらは本講義で学ぶ内容を箇条書きにしたもの．）

1. アルゴリズム記述のための言語と基本的制御構造
2. 基本的データ構造（１）（型、配列、レコード）
3. 基本的データ構造（２）（列、バッファ）
4. 探索アルゴリズム、計算量
5. 整列アルゴリズム（１）（単純挿入整列、単純選択整列、単純交換整列）
6. 整列アルゴリズム（２）（ヒープ整列、クイック整列、マージ整列）
7. 再帰的アルゴリズム（１）（ハノイの塔）
8. 再帰的アルゴリズム（２）（再帰曲線）
9. バックトラックアルゴリズム（１）（8 王妃問題、ナイト巡回問題）
10. バックトラックアルゴリズム（２）（ナップザック問題、ＴＳＰ）
11. 動的データ構造（１）（線形リスト）
12. 動的データ構造（２）（木構造）
13. 動的データ構造（３）（木構造の操作）
14. ハッシング
15. その他のアルゴリズム
16. 定期試験

（３）教科書・参考書

教科書：N.ヴィルト（浦昭二、國府方久訳）「アルゴリズムとデータ構造」、近代科学社、1990

参考書：C言語によるアルゴリズム記述参考書：柴田望洋、辻亮介「C言語によるアルゴリズムとデータ構造」 ソフトバンクパブリッシング、2002

※ なお，本講義では，毎回配布する資料を中心にして解説しますので，教科書・参考書は必ずしも購入しなくても構いません．

（４）履修目標

1. 基本的データ構造を理解し、アルゴリズム記述に利用できる。
2. 探索アルゴリズムを理解し、その計算量の評価法を他者に説明できる。
3. 整列アルゴリズムを理解し、その計算量を解析して他者に説明できる。
4. 再帰的アルゴリズムを理解し、その動作原理を他者に説明できる。
5. バックトラックアルゴリズムを理解し、応用問題に適用できる。
6. 動的データ構造を理解し、応用問題に適用できる。
7. ハッシングを理解し、その計算量について解析できる。

（５）履修条件および関連科目

本講義で習得した基本的アルゴリズムを、C言語のプログラムとして記述するために**「プログラミング」**がある。また、C言語を用いて実際の問題にアルゴリズムの適用する演習を**「プログラミング演習Ｉ」**で行う。

なお，これらの講義と関連しますが完全に連動してはいないことにご注意下さい．

（６）成績の評価

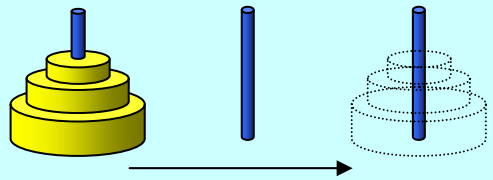
講義中の小テスト、レポート、中間テスト、および期末試験などを総合的に評価する。

3. 平成 24 年度講義（予定）

1. 基本的データ型
2. 基本的制御構造
3. 変数のスコープルール．関数
4. 配列を扱うアルゴリズムの基礎(1)．最大値，最小値
5. 配列を扱うアルゴリズムの基礎(2)．重複除去，集合演算，ポインタ
6. ファイルの扱い
7. 整列(1)．単純挿入整列，クイック整列
8. 整列(2)．マージ整列
9. 再帰的アルゴリズムの基礎．再帰におけるスコープ．ハノイの塔など．
10. バックトラックアルゴリズム．8 王妃問題など．
11. 線形リストを扱うアルゴリズム(1 回)
12. 木構造を扱うアルゴリズム(1) 基礎
13. 木構造を扱うアルゴリズム(2) 挿入，削除，バランスなど．
14. ハッシング
15. その他のアルゴリズム

前半は1年生用講義「プログラミング入門」の内容の復習的な意味合いもあります

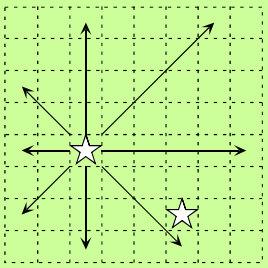
（例１）ハノイの塔



円盤を移動する最小手数を求める問題

（例２） ８王妃問題

チェスの王妃（上下左右斜め方向どこまでも移動可能）を，8×8 マスに 8 個，互いに他を取り合わない関係の位置に配置する最適配置問題．



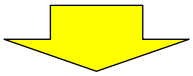
☆：チェスの王妃

一見すると非常に複雑そうな問題を，コンピュータを使ってどのように解くのかについて考えることで，情報工学的に問題を解決する能力を高めることが目標です．楽しいですよ！

雑談ノート： 本講義に対応する旧講義「情報基礎論 IA・IB」を受講した皆さんの先輩学生の意見の紹介と，それに対する担当教員の私見

ある先輩の意見：

『与えられた課題のアルゴリズムを考えてプログラムを書くことの意義が良くわかりません．例えば，パズルの一種であるハノイの塔について考えて解くことが，今後，卒業研究や企業での仕事で役に立つとは思えないのですが...』



これに対する担当教員の回答：

おっしゃる通り，本講義で扱う問題はいわゆる**「問題のための問題」**とでも呼ぶべきものが多く，実用的な問題ではないかも知れません．しかしながら，本講義および本講義に関連するプログラミングの演習を通して，皆さんが次のことを勉強できれば良いと思っています．

1. 問題を順序だてて解く（すなわちアルゴリズムを組み立てる）能力とその習慣．
2. 考えたアルゴリズムをプログラムに置き換えることができること，すなわちプログラミング能力の上達．
3. バグ（プログラムのミス）が出たときの対処方法，デバッグ（プログラムを修正すること）の精神的労苦に耐える忍耐力と，長時間，物事に集中する力．
4. 目的のもの（正解のプログラム）を完成することが出来たとき，言い換えれば，プログラムが自分の思い通りに動作したときの達成感と喜び．

これらはいずれも，将来，皆さんが卒業研究で実際に行う研究や，就職後の職場で行う仕事で必ず必要になる素質・能力です．ですから，

『 本講義（と関連するプログラミングの演習）の目的は，課題を解くことそれ自身ではなく，むしろそのプロセスで皆さんが経験するであろう貴重な上記のようなことがらである． 』

と言えるでしょう．あくまで私見ですが...

本講義（・演習）で苦労して獲得した能力は，いずれ皆さん自身にとって大変役立ちます．卒業研究で何かの問題や計算を解くためのプログラムを作成しなければならなくなったときも，本講義を履修していない人に比べて格段の差で素早く行うことができるでしょう．

それでは，次頁から今回の座学の本題（基本的データ構造）です．

【アルゴリズム序論 --- 基本的データ型】

☆ アルゴリズムとデータ構造

アルゴリズム	+	データ構造	=	プログラム	
(Algorithms)		(Data Structures)		(Programs)	--Niklaus Wirth

参考図書

N.Wirth（浦 他訳）「アルゴリズムとデータ構造」, 近代科学社 (3,800 円)
L.Hancock,M.Krieger（三浦 訳）「C 言語入門」,アスキー出版 (2,580 円)
B.Kernighan,D.Ritchie(石田 訳)「プログラミング言語 C 第 2 版」,共立出版 (2,800 円) ← 定番！

ただし中級者向き

初学者用 参考図書（講義の一部しかカバーしていません）

林晴比古著「新 C 言語入門ビギナー編」,ソフトバンク (1,900 円)
柴田望洋著「明解 C 言語 入門編」,ソフトバンク (2,500 円)
奥村晴彦著「C 言語による最新アルゴリズム事典」,技術評論社 (2,330 円)

◎アルゴリズム

・ある問題を解くための計算方法.
・機械的に実行でき，有限時間内に必ず答を出して終了する計算方法.
・良否の基準
　効率の良さ(実行時間，使用記憶領域量)，解の安定性，誤差，プログラムの簡単さ

◎データ構造

・現実の状況をどのような方法で抽象化するか
・どのように表現するか

☆プログラム言語（プログラミング言語）

・計算機の内部表現は 2 進数
・プログラム言語による隠蔽　→　言語に用意される基本データ型(とその組合わせ)

プログラム言語 → 「データ構造とその演算」,「制御構造」を定義

※この講義では C 言語を使う.

☆データ型の概念

プログラム言語内の“値”を持つものすべてはある一つの型を持つ

　定数，変数，式，関数

型 … あるデータ(2 進数)を何に解釈するかを表したもの

　データの解釈　+　データの大きさ

構造を持たない型(基底型)　　分解できない一つの値の型
構造を持つ型　　複数の(ある型に属す)値から構成される値の型

◎基本データ型

		一般的な 32bit マシンでは
char	1 文字(英数字)	8bit
int	整数	32bit
float	単精度浮動小数点数	32bit
double	倍精度浮動小数点数	64bit

bit とは 2 進数の 0 か 1 のこと. 例えば char は 8 桁の 2 進数で表現され，値としては 0～255 をもつ．以下同様

・char を除いてどれだけの大きさの記憶が割り当てられるかは計算機/言語処理系に依存
・short / long (int, double), signed / unsigned (char, 整数)などの修飾子がある
・真偽値は通常 int を用いて表す．真 ⟷ 0 以外，偽 ⟷ 0

◎構造を持つ型

・配列（array）：同じ型をもつデータのまとまり

例えば，

> クラス 80 名の試験の点数を入力して，点数が高い順に並べ直す

ことを行うとします．
このとき，まず学籍番号順に試験の点数を入力してからソーティング（データの並べ替え）を行う必要があるでしょう．では，まず点数をどのような**変数**に代入すればよいでしょうか？

80 個の変数（seiseki1, seiseki2, seiseki3, …, seiseki80）を用意して，キーボードから入力するとすると，まず，変数の宣言部で 80 個の変数を宣言する必要があります．

```
int seiseki1, seiseki2, seiseki3, ……, seiseki80;
```

そして，入力するときは，

```
printf("学籍番号 1 の人の点数 = ");   scanf("%d",&seiseki1);   printf("¥n");
printf("学籍番号 2 の人の点数 = ");   scanf("%d",&seiseki2);   printf("¥n");
  ・・・・・・・・・・・・・・・・・・
```

としなければなりません．これはあまりに非効率的で，プログラムも醜くなります．
そこで**配列**が必要になります．
例えば 80 個の整数型の配列 seiseki[ ] は次のようにして使用宣言します．

```
int seiseki[80];
```

これによって記憶領域に整数型変数 seiseki[0], seiseki[1], seiseki[2], …, seiseki[79] が確保されます．そしてデータ入力後は，例えば次のようにするだけでデータを表示することも可能となります．

```
for ( i = 0; i < 80; i ++ ){
    printf("学籍番号 %d の人の点数 = %d¥n", i, seiseki[i]);
}
```

どうですか？　配列の必要性を理解することができましたか？
配列は２次元にすることも可能です．例えば，横 512 個×縦 512 個の unsigned char 型の２次元配列は次のように宣言します．その後，image[100][100]などのように特定の要素の値を取り出すことが可能です．

```
unsigned char image[512][512];    /* 画像を表すときなどに使います */
```

・構造体（struct）：異なる型をもつデータのまとまり

例えば，

> クラス 80 名の優・良・可の数，それらから計算した平均点のデータを入力して取り扱う

とします．
ここで，**優・良・可の個数は整数型，平均点は浮動小数点型**で扱うとします．先ほどの配列は全て同じ型のデータを並べたものでした．**構造体**は，**異なる型のデータのまとまり**を扱うためのものです．

このとき，これらをひとまとまりとする新しい型（構造体）を宣言するには，例えば

```
struct seiseki {
    int num_of_A;
    int num_of_B;
    int num_of_C;
    double average;
};
```

というようにします．

ここで，seiseki は構造体に対して自由に付けた名前で，**構造体タグ**と呼ばれます．また，それを構成している変数群を**メンバー**と呼びます．これにより「struct seiseki 型」という型をプログラム中で使えるようになります．

そこで，この「struct seiseki 型」の変数 student を宣言するときは次のようにします．

```
struct seiseki student;
```

この変数 student のメンバーは（**変数名**）.（**メンバー**）で表されます．具体的には，次のような代入などが可能になります．

```
student.num_of_A = 40;
student.num_of_B = 10;
student.num_of_C = 0;
student.average = 95.5;
```

ピリオド

このような変数 student が 80 個集まった配列 fm を考えることもできます．すなわち，**構造体の配列**です．このときは，struct seiseki の定義の後で，次のような配列の使用宣言を行います．

```
struct seiseki fm[80];
```

これにより，例えば学籍番号 i の人の平均点に対する次のような代入などが可能になります．

```
fm[i-1].average = 80.0;
```

◎動的データのための型

・**ポインタ（pointer）：変数が格納されている記憶領域のアドレス（番地）を指すもの**

ポインタの扱いの習熟はＣ言語攻略のキーポイントですが，取り扱いはやや複雑です．本講義の４～５回目あたりで詳しくやることになるでしょう．

☆定数

基本的に数字，小数点，負符合，指数表示を組み合わせて表現

123	int の定数
0.123	double の定数
1.23e-1	同上（1.23×10^{-1}）
'a'	文字 a

0 と 0.0 の違い：
0 は整数，0.0 は浮動小数点とみなされます．

☆記号定数

#define　名前　文字列

・「名前」が引用符なしで現れる場合，「文字列」に置き換わる．
　マクロ機能
・「名前」は大文字で書くのが慣習
・例

```
#define N 10
```

なぜ必要か？

・　何を表しているかが一見して解らない定数である**マジックナンバー**をプログラム中に残さないため．
　例）・・・・・・
　　n = 1.2345;　←　これって何の値？
　・・・・・・

・　値を変更するときに楽．さもないとその変数（定数）が出てくるところ全てを変更する必要がある．

これ以降，すべての N は 10 で置き換えられる．

※ **式と違うので ； はつけないこと**．つけると，それを含めて置換えがされる．

☆変数

・値を保持する名前のついた記憶領域
・変数の型 → どの型の値を保持するかを示す
・英文字で始まる文字列を名前とする

☆変数の宣言と初期化

・変数は使用する前に宣言(=~ 定義)
　型 変数(名), 変数, ..., 変数 ;

・初期値を = に続けて書ける.

```
int    x;
int    y = 10;
double    z;
```

※ 直感的には，変数の宣言により記憶領域が割り当てられる.

☆基本演算

式を構成する.

・算術演算
　+, -, *, /,　%(余り)

例）10 * n,　1 + 2 + k,　など

・関係演算子と論理演算子
　条件判断などに使う
　真(1)か偽(0)が値となる

・等値演算子
　= = (等しい),　! = (等しくない)

例）if （a = = b） c = 1; など

・関係演算子
　>,　>=,　<,　<=

例）if （a > 0） c = 1; など

・論理演算子
　&& (かつ),　|| (または)

例）if （a > 0 && b > 0） c = 1; など

・代入 = (演算とはすこし異なる)
　変数の値の再定義

例）a = 1;　　a = b;　など

　変数 = 式
　(式の値を変数に設定)

例）a = a + 1;　　a = b + c;　など

右辺の計算結果を左辺の変数の新しい値として代入せよ，の意．その変数が右辺にも現れるときは，その変数の以前の値を指す.

次頁から，今回に関係するサンプルプログラムを示します.

```
/*****************************************************************

      アルゴリズムとデータ構造
      サンプルプログラム  basictypes.c
      <<基本データ型と変数>>

      copyright (c) 1995,96,97    T.Mori <mori@forest.dnj.ynu.ac.jp>
 *****************************************************************/

/* ←これらの記号で囲まれた部分はコメント→ */

/* 入出力関連の定義の読み込みをコンパイラに指示 */
#include <stdio.h>

/* 記号定数の定義 */
#define W 3.0
#define MUL c*d

/*
 * C プログラムは必ず，`main'という名前の「関数」から実行される．
 * そのため，まず，main という名前の関数を定義しなければならない．
 * 以下のプログラムはその例です．
 * 関数名`main'の後ろにある括弧は`main'への引数(ひきすう)を表しますが，
 * この例では引数はありません．
 *
 */

main( )
{
   /* 整数型(int)の変数 x,a,b を使うことを宣言 */
   /* 変数 b は値 2 に初期化される */
   int     x, a, b = 2;
   /* 浮動小数点型 */
   float   y, c, d;
   /* 文字型 */
   char s, t, u;

   /* 整数の演算 */
   a = 3;
   /* 変数 a に 値 3 を代入する．
    * その結果，変数 a の値は 3 になる．
    */

   printf("a = %d¥n", a);
   printf("b = %d¥n", b);
   /* 変数 a, b の値を表示する．
    * printf は引数(第一引数)に与えられた文字列を表示するが，
    * 文字列の中に %d が現われると次の引数の値が整数型であると仮定し，
    * %d をその値で置き換えた文字列を表示する．
    * また，'¥n'は特殊文字の一つである「改行」を表している．
    */
   x = a + b;
   /* 変数 a の値と b の値を加え，その値を 変数 x に代入する．*/

```

1 番左の数字は説明のための行番号で，プログラムには含まれません．

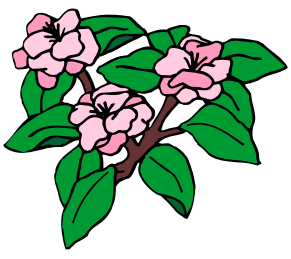
```
55      printf("a + b == %d¥n", x);
56
57      x = a - b;
58      /* 変数 a の値から b の値を引き，その値を 変数 x に代入する．*/
59      printf("a - b == %d¥n", x);
60
61      x = b - a;
62      printf("b - a == %d¥n", x);
63
64      x = a * b;
65      /* 変数 a の値と b の値をかけ，その値を 変数 x に代入する．*/
66      printf("a * b == %d¥n", x);
67
68      x = a / b;
69      /* 変数 a の値を b の値 でわり，その値を 変数 x に代入する．
70       * ただし，整数演算なので，端数は切捨てられる．
71       */
72      printf("a / b == %d¥n", x);
73
74      x = a % b;
75      /* 変数 a の値を b の値 でわったあまりを変数 x に代入する．*/
76      printf("a %% b == %d¥n", x);
77      /* printf の文字列中の `%%' は `%'自身を表示することを意味する．*/
78
79      x = (a == b);
80      /* 変数 a と 変数 b の等値性を判定し，その真偽値を変数 x に代入する．*/
81      printf("(a == b) == %d¥n", x);
82
83      x = (a == b || a != b);
84      /*「a == b または a != b である」を判定し，その真偽値を変数 x に代入する．*/
85      printf("(a == b || a != b) == %d¥n", x);
86
87
88      /* 浮動小数点数の演算 */
89      c = 3.0;
90      d = 0.02e2; /* 指数表示　0.02 x (10 の 2 乗) */
91
92      printf("c = %f¥n", c);
93      printf("d = %f¥n", d);
94      /* printf の文字列の中に %f が現われると対応する引数の値が
95       * 浮動小数点型であると仮定し，%f をその値で置き換えた文字列を表示する．
96       */
97      y = c + d;
98      printf("c + d == %f¥n", y);
99
100     y = c - d;
101     printf("c - d == %f¥n", y);
102
103     y = d - c;
104     printf("d - c == %f¥n", y);
105
106     y = MUL;
107     printf("c * d == %f¥n", y);
108
109     y = c / d;
110     printf("c / d == %f¥n", y);
```

```
111
112     x = (c == d);
113     printf("(c == d) == %d¥n", x);
114
115     x = (c == d || c != d);
116     printf("(c == d || c != d) == %d¥n", x);
117
118     /*  文字型の演算  */
119     /*  文字型は 8bit の整数データ(=  文字コード)と同じである  */
120     s = 'g';     /* `g'という文字が値  */
121     t = 'h';     /* `h'という文字が値  */
122     printf("s == %c¥n", s);
123     printf("t == %c¥n", t);
124
125     x = (s < t); /*  関係演算子で順序がわかる  */
126     printf("(s < t) == %d¥n", x);
127
128     x = (s > t);
129     printf("(s > t) == %d¥n", x);
130
131     u = s - 1; /* s の前の文字  */
132     printf("(s-1) == %c¥n", u);
133 }
```

【実行結果】

```
a = 3
b = 2
a + b == 5
a - b == 1
b - a == -1
a * b == 6
a / b ==
a % b ==
(a == b) == 0
(a == b || a != b) ==
c = 3.000000
d = 2.000000
c + d == 5.000000
c - d == 1.000000
d - c == -1.000000
c * d == 6.000000
c / d ==
(c == d) == 0
(c == d || c != d) ==
s == g
t == h
(s < t) == 1
(s > t) == 0
(s-1) == f
```

```
/*****************************************************************

        アルゴリズムとデータ構造
        サンプルプログラム  compoundtypes.c
        <<構造を持つ型と変数>>

        copyright (c) 1995,96,97    T.Mori <mori@forest.dnj.ynu.ac.jp>
 *****************************************************************/

#include <stdio.h>


main( )
{
    /* 3 個の int 型データを保持する配列 a の宣言．a[0]〜a[2] が使える */
    int a[3];
    /*
     * 構造体 point の宣言．x,y という名前をもつ二つの int 型のメンバを持つ．
     * 構造体の宣言は新しい型の定義となる．
     */
    struct point {
        int x;
        int y;
    };
    /*
     *   構造体タグ point で示される型(「struct point 型」という)の変数 pt の宣言．
     */
    struct point pt;
    /*
     * 構造体 item の宣言．key(int 型),data(float 型)という名前のメンバをもつ．
     */
    struct item {
        int key;
        float data;
    };
    /*
     *   struct item 型の変数 d,e の宣言，初期化．
     *   初期化によって，e.key == 5, e.data == 1.8 となる．
     */
    struct item d;
    struct item e = {5, 1.8};
    /**/
    int i;

    /*
     * 配列の添字は，整数式であればなんでもよい．
     */
    a[0] = 1;
    i = 0;
    a[i+1] = 2;
    a[i+2] = 3;

    printf("a[0]==%d, a[1]==%d, a[2]==%d¥n", a[0],a[1],a[2]);

    /*
     * 構造体のメンバは`.'(ピリオド)を使って示す．
```



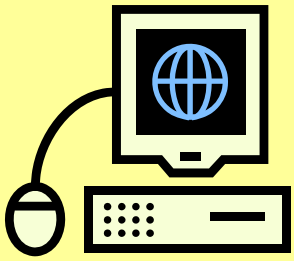
```
57        * 変数と同様に扱える.
58        */
59      pt.x = 1;
60      pt.y = pt.x + 3;
61      printf("pt.x==%d, pt.y==%d¥n", pt.x, pt.y);
62
63      /*
64        * 構造体を型に持つ変数の代入.
65        * 一括して値をコピーすることができる.
66        */
67      d = e;
68      printf("d.key==%d, d.data==%f¥n", d.key, d.data);
69  }
70
```

【実行結果】

```
72      a[0]==1, a[1]==2, a[2]==3
73      pt.x==1, pt.y==4
74      d.key==5, d.data==1.800000
```

本講義で配布する資料(pdf 版)は，次のページからダウンロード可能です（順次更新されます).

http://www.tommylab.ynu.ac.jp/lecture/Algorithm/



ではまた来週１時限にお会いしましょう