

情報システムプログラミングⅡ (**10**回目)

2024年6月21日 (金)

3～4限

授業内容

- 講義内容（教科書の316～345ページ）
 - C言語の特徴
 - メモリ
 - アドレスの取得
 - アドレスの解決
- 演習課題

C言語の特徴

■低水準（低級）言語

- 機械語やアセンブリ言語などのプログラミング言語の総称
- コンピュータ寄り，高水準言語の対義語

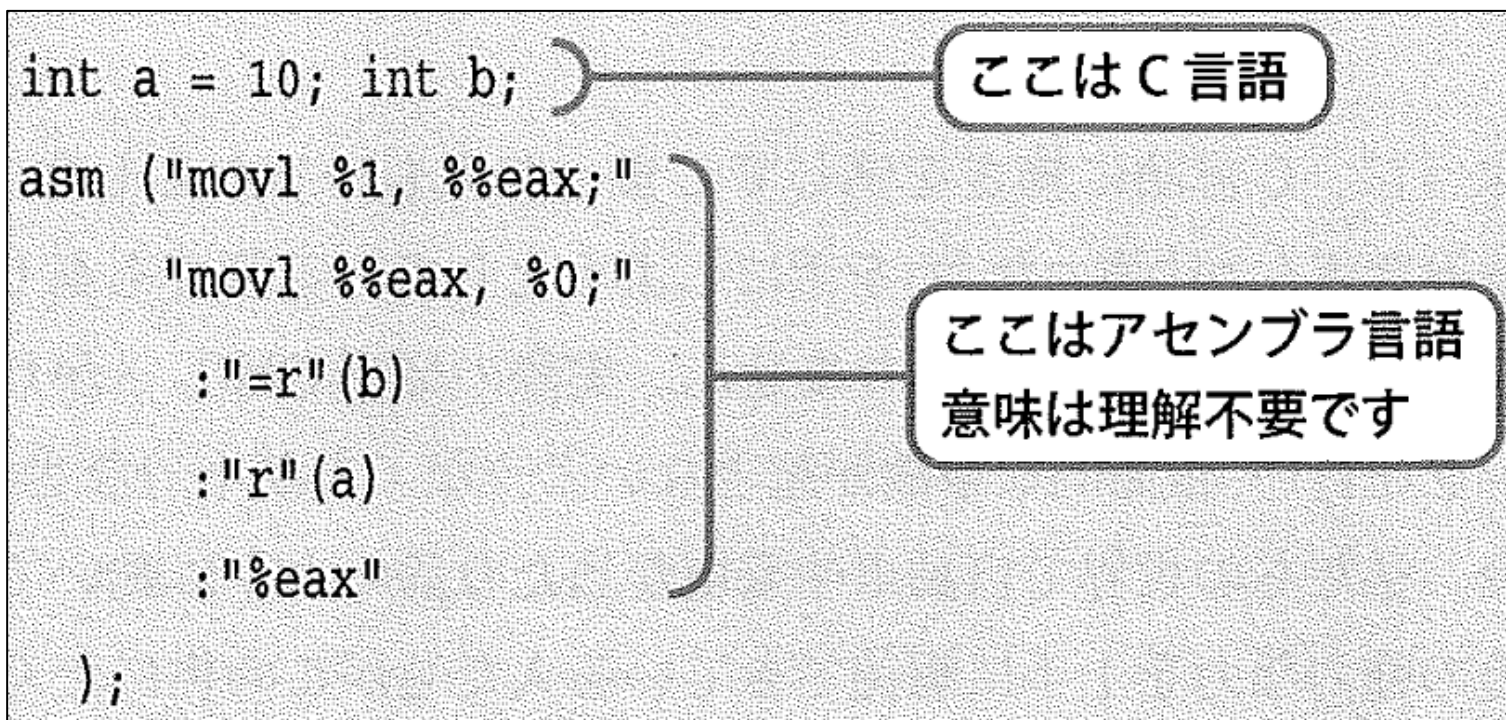
■高水準（高級）言語

- 記述の抽象度が高いプログラミング言語の総称
 - 具体的にはPythonやJavaScriptなど
 - **C言語は低水準言語の特徴を持つ高水準言語**
- コンピュータの利用者（人間）寄り，低水準言語の対義語

C言語の特徴

■ インラインアセンブラ

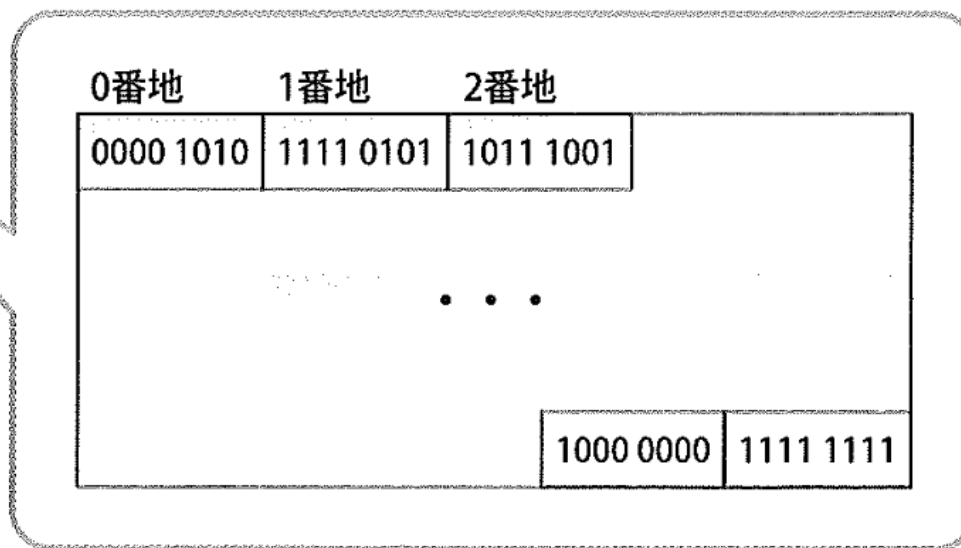
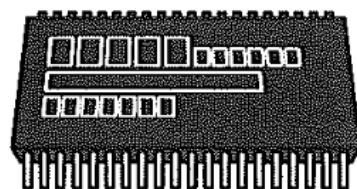
- アセンブリ言語をC言語のプログラムに書き込むことができる



メモリ

■メモリとは

- コンピュータ内で一時的に情報を記憶する装置（主記憶装置）
- メモリ内での情報の格納場所は「アドレス」といい、一意の数値で表される（単位は番地）



メモリ

■メモリの利用

- プログラムは実行時，メモリを利用して一時的に情報を記憶する
- 高水準言語では，例えば以下のようなメモリに関する処理が自動的に行われている
 - 変数定義：メモリ上の領域（メモリ領域）を確保する
 - 代入：メモリ上の指定アドレスに情報を書き込む
 - 取得：メモリ上の指定アドレスから情報を読み出す
- メモリ領域は主に静的領域，スタック領域，ヒープ領域の3つの区分で分けられ利用される

アドレスの取得

■対象が格納されているアドレスの取得

- アドレス演算子である「&」を利用する
- アドレス演算子によりアドレスを取得する構文

& 変数

※変数が確保されているメモリ領域の先頭番地に「化ける」。

```
int main(void)
{
    int a = 70;
    printf("変数aには70が入っています\n");

    long addrA = (long)&a;
    printf("変数aのアドレス: %ld\n", addrA);
    return 0;
}
```

変数 a のアドレスを取得して代入 (long 型に入れるために要キャスト)

long 型のプレースホルダ

アドレスを取得したい対象の先頭に「&」を付ける

実行結果

変数aには70が入っています

変数aのアドレス: 3921

実行するたびに値は変わる

アドレスの取得

■scanf関数の仕様

- 第2引数以降にはアドレスを指定する必要がある
- おまじないで付けていた「&」はアドレスを取得するため

```
int a; unsigned int b; double c; char d; char e[11];  
scanf("%d %u %lf %c %s", &a, &b, &c, &d, e);  
printf("a:%d b:%u c:%f d:%c e:%s", a, b, c, d, e);
```


配列名に「&」が不要な理由は
次回以降の授業で・・・

アドレスの取得

■ メモリ領域の確保

- 対象を格納するために必要な領域（大きさ分）が確保される

`int a = 70;` → 4バイトの整数を入れる箱aを作りたい!



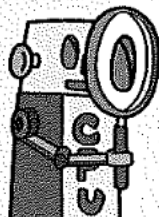
空いている場所を探そう。
どれどれ…。
よし、3921～3924を使うことにしよう。

```
0100011011101000101110101001
1010101111110001000000010111
1101100010001101010101111000
1100011000011101110001010011
11
1011100001011011101100
```

3921～3924番地

「&」で取得できるのは
先頭のアドレスのみ
(この場合は3921)

`addrA = &a;` → 変数aのアドレスを知りたい!



ええと、変数aはどこに作ったかな?
あ、ここだ。
はい、先頭は3921番地です。

```
0100011011101000101110101001
1010101111110001000000010111
1101100010001101010101111000
1100011000011101110001010011
1101000110000000000000000000
0000001011100001011011101100
```

3921～3924番地 (変数aが確保済み)

アドレスの取得

■ アドレスを格納するための型（ポインタ型）

- どのようなアドレスでも格納できる型として「void*」型がある

➤ 「void」型とは別物なので注意！

- void*型の変数を定義するための構文

void* 変数名;

アドレスを格納する変数を
ポインタ変数という

アドレスを扱うための
プレースホルダは「%p」

```
int main(void)
{
    int a = 70;
    printf("変数aには70が入っています\n");

    void* addrA = (void*)&a;
    printf("変数aのアドレス: %p\n", addrA);
    return 0;
}
```

void* 型に a のアドレスを代入

アドレス専用のプレースホルダ

アドレスの解決

■ 指定アドレスからの情報の取り出し

- 間接演算子（間接参照演算子）である「*」を使う
- 間接演算子によりアドレスに格納された情報を取得する構文

* ポインタ変数名



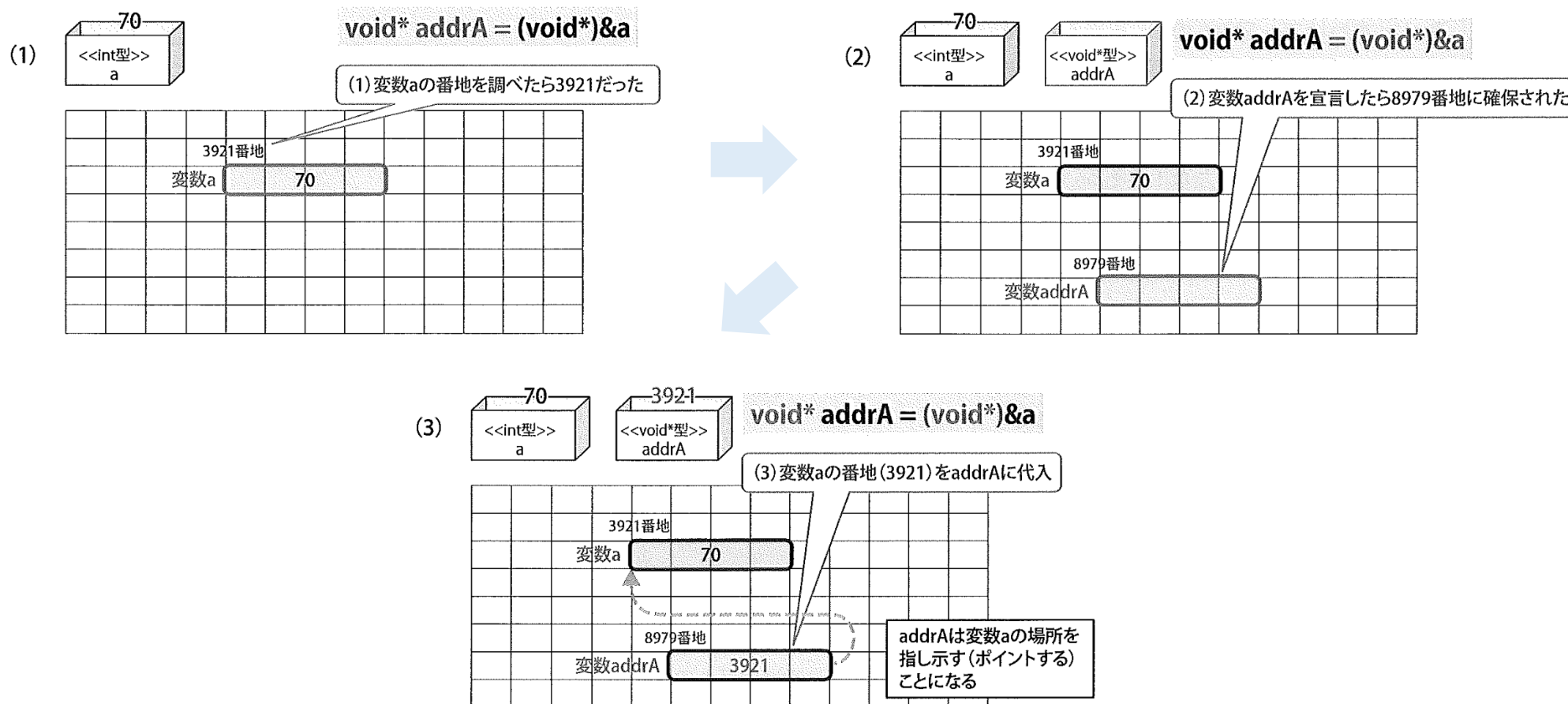
```
int main(void)
{
    int a = 70; ①
    printf("変数aには70が入っています\n");

    void* addrA = (void*)&a; ②
    printf("変数aのアドレス: %p\n", addrA);
    printf("%p番地に格納されている情報: %d\n", addrA, *addrA); ③

    return 0;
}
```

アドレスの解決

■ 指定アドレスからの情報の取り出し



アドレスの解決

■実用的なポインタ型

- 既存の型の末尾に「*」を付けると、その型に対応したアドレスのみを格納できるポインタ型（ポインタ変数）を定義できる
 - 例えば「int*」型、「double*」型、「char*」型など
- int*型の場合、この型の変数に
 - 格納されているアドレスは、int型の変数の先頭のアドレスであると認識される
 - 間接演算子（*）を用いると、格納されているアドレスからint型の分（4バイト分）の情報が取り出される

アドレスの解決

■実用的なポインタ型

- 基本的には各型に対応したポインタ型を利用すれば良い

```
int main(void)
{
    int a = 70;
    printf("変数aには70が入っています¥n");

    int* addrA = &a;
    printf("変数aのアドレス: %p¥n", addrA);
    printf("%p番地に格納されている情報: %d¥n", addrA, *addrA);

    return 0;
}
```

int* 型に a のアドレスを代入

addrA 番地から 4 バイト分のメモリの内容を取り出す

キャスト（型変換）は不要

実行結果

変数aには70が入っています

変数aのアドレス: 3921

3921番地に格納されている情報: 70

実行するたびに値は変わる

アドレスの解決

■間接演算子の取り扱い

- メモリ上のどこでもアクセスできるため、取り扱いに注意すること（不用意に利用しないこと）！

```
// メモリ0～3番地の内容を表示するには…
```

```
int* p = 0;  
printf("%d", *p );
```

0 番地を先頭とした int 型変数に化ける

```
// メモリ9410～9411番地に794を書き込むには…
```

```
short* q = 9410;  
*q = 794;
```

9410 番地を先頭とした short 型変数に化ける

このコードを実行すると PC 環境を破壊する恐れがあるため、試しにであっても動かさないでください。理由は第 10 章で紹介します (p.375)。

アドレスの解決

■ 「*」 記号の取り扱い

- ①と②の「*」は別物なので注意
- ①は③と書くこともできるが基本的に①で書くこと

```
int* addrA;           --①
printf("%d", *addrA); --②
int *addrA;           --③ (①の伝統的な書き方)
```

■ ポインタ変数の定義

- 複数のポインタ変数をまとめて定義できないので注意

```
int *a, *b;           // 正しい。int*型のaとint*型のbを宣言
```

```
int* a, b;            // 間違い。int*型のaとint型のbを宣言
```