応用プログラミング

第6回　配列の受け渡し、ポインタ

メディア情報コース
平居　悠（ひらい　ゆたか）

到達目標

- C言語を通じてコンピュータの根幹部分を理解する。
- 目的に応じたプログラムを自由に設計できる。

前回

第１回　C言語の基本文法・言語処理系
第２回　数学的計算、配列、文字列
第３回　型変換、入出力処理
第４回　デバッグ、関数定義、変数の分離
第５回　プロトタイプ宣言、分割コンパイル
第６回　配列の受け渡し、ポインタ
第７回　まとめ、期末試験

前回の目標

- 関数をプロトタイプ宣言して読みやすいコードを書けるようになる。
- 複数のソースファイルに書かれたプログラムを分割コンパイルできるようになる。

前回学んだこと

1. プロトタイプ宣言
2. 分割コンパイル

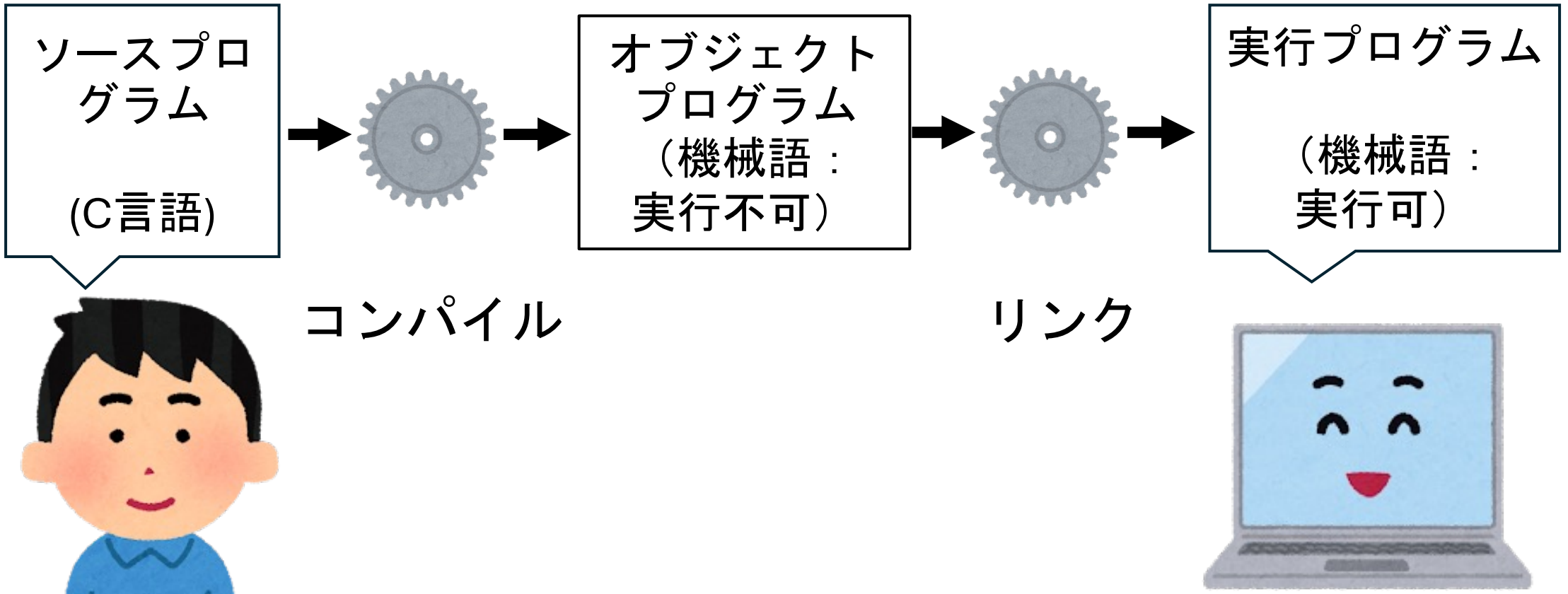
プロトタイプ宣言

後で利用する関数について、その引数の個数と型、返す結果の型を先に宣言すること。

プロトタイプ宣言

```
#include <stdio.h>

int factorial(int);

int main()
{
   int i;
   for (i = 0; i < 11; i++)
      printf("%d! = %d\n", i, factorial(i));

   return 0;
}

int factorial(int n)
{
   if (n == 0)
      return 1;
   else
      return n * factorial(n-1);
}
```

機械語への変換処理：実際の行程



リンク

プログラムの実行に必要な処理やprintfやscanfなどのシステムで用意された関数の実体をオブジェクトプログラムと結合すること

分割コンパイル

以下のようにターミナルに入力する。

```
gcc -c main.c
gcc -c sub.c
gcc -o main main.o sub.o
```

makeユーティリティ

ソースファイルが多くなると、コンパイル＆リンク作業を行うのが複雑になってくる。makeユーティリティはこの負担を軽減してくれる。

Makefile

makeにファイル依存関係を教えておくと、makeを起動したときに適切な処理を行ってくれる。makeに依存関係を教えるには、作業ディレクトリに**Makefile**というファイルを置き、そこに依存関係を記述する。

Makefileの書き方

ターゲット：<TAB空白>依存ファイル（群）
<TAB空白＞それを作るためのコマンドライン

- コロン(:)の後とコマンドラインを各行の先頭はスペースではなく**TAB**コードを入れる。
- #で始まる行はコメント行として解釈対象外となる。
- 慣習として最初にallというターゲットを置き、最終ターゲットをその右辺に書く。

makeの起動

make ターゲット

makeは指定されたターゲットとその右辺に書かれた依存ファイル(群)のどちらが新しいファイルか調べ、依存ファイルの方が新しい場合、ターゲットも更新が必要だと判断し、次の行から始まるコマンドラインを起動する。

前回の問い

- プロトタイプ宣言とは何か？
  - 後で利用する関数について、その引数の個数と型、返す結果の型を先に宣言すること。
- 分割コンパイルをするにはどのようにすれば良いのか？
  - “gcc –c”でオブジェクトファイルを作成し、リンクする

今回の目標

「ポインタ」の概念を理解し、
使いこなせるようになる。

今回学ぶこと

1. メモリのアドレス
2. ポインタ
3. 配列とポインタ
4. 配列の受け渡し

今回の問い

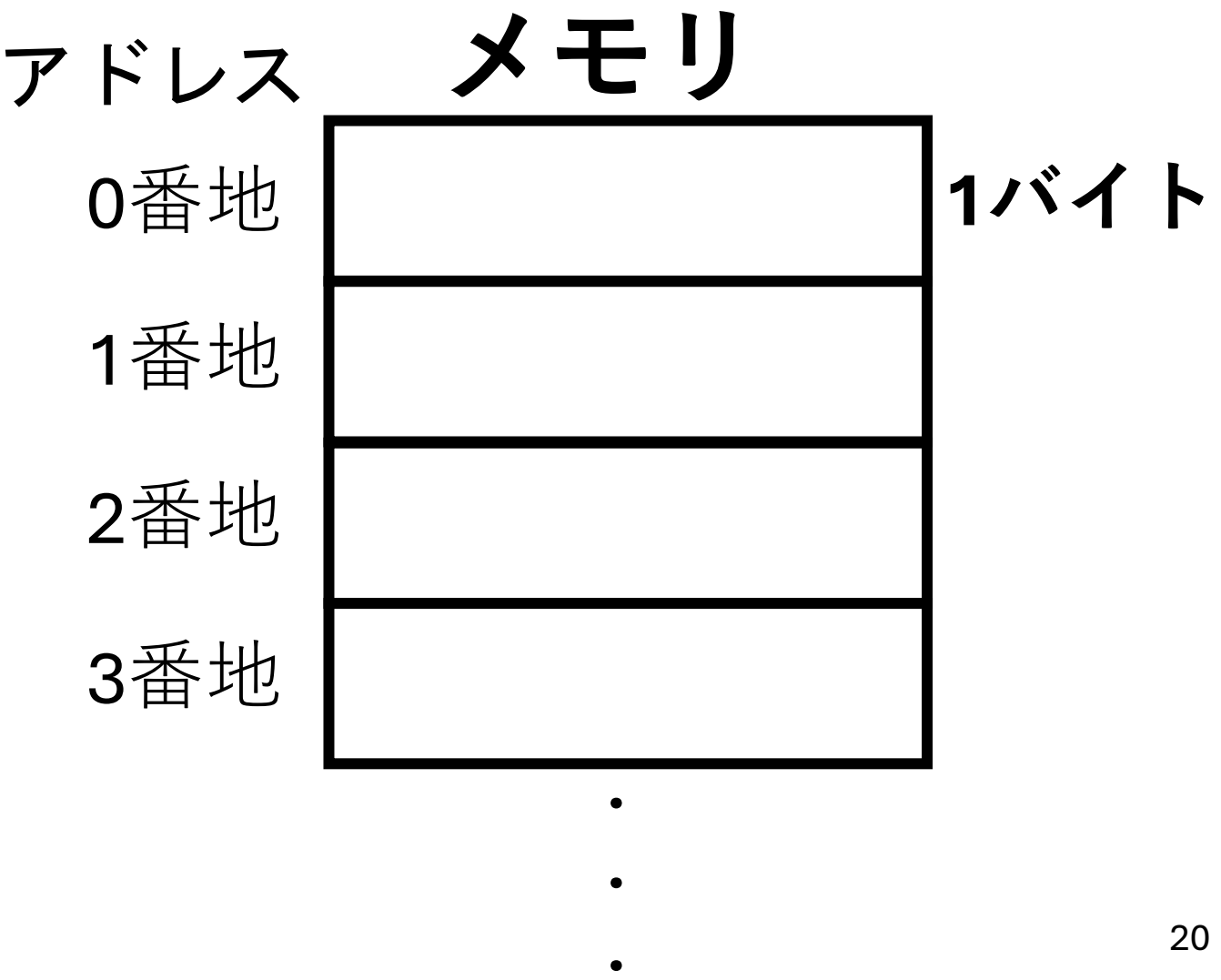
- ポインタとは何か？
- 関数に配列を渡すにはどのようにすれば良いのか？

今回学ぶこと

1. メモリのアドレス
2. ポインタ
3. 配列とポインタ
4. 配列の受け渡し

メモリのアドレス

- メモリは1バイトずつ区分けされており、これらには0番地、1番地、2番地...のように独自の住所が割り当てられている。
- この住所のことを**アドレス**という。



アドレス演算子 (&)

変数に「**& (アンパサンド**）」を付けるとその変数が保存されているメモリのアドレスを参照できる。

&変数名

例えば「&a」は、変数aが保存されているアドレスを意味する。

今回学ぶこと

1. メモリのアドレス
2. **ポインタ**
3. 配列とポインタ
4. 配列の受け渡し

ポインタとは

アドレスを格納するための変数

どんな時にポインタを使うか？

- 関数から複数の値を返してもらう。
- 配列にアクセスする。
- 連結リストや木構造などのデータ構造を表現する。

ポインタの宣言

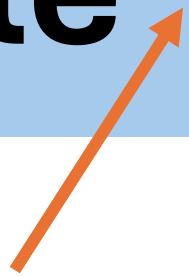
int型へのポインタ

```
int *pi;
```

ポインタの宣言

double型へのポインタ

```
double *pd;
```

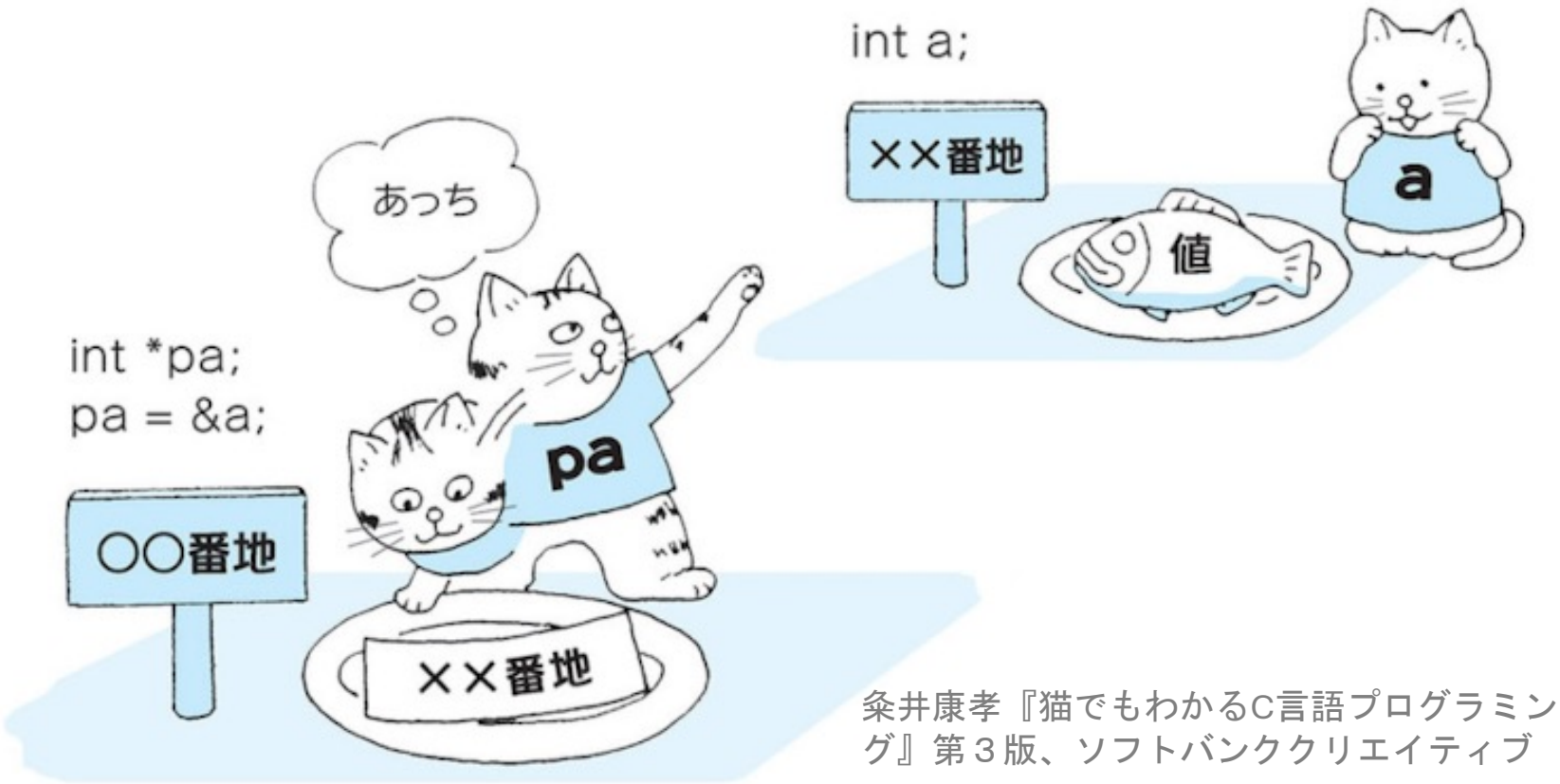


ポインタ宣言子

アドレスの代入

変数aのアドレスをポインタpaに代入する場合：

```
int a;
int *pa;
pa = &a;
```



粂井康孝『猫でもわかるC言語プログラミング』第３版、ソフトバンククリエイティブ

paの値を見れば、aの番地がわかる

アドレスの代入

変数に値を代入する場合：

```
int a, b;
int *pa;


a = 5;      /* 変数aに5を代入 */
pa = &a;  /* ポインタpaにaのアドレスを代入*/
b = *pa;  /* bに5が代入される */
```

間接参照演算子

ponter01.cというファイルを作成し、右のコードを書いて実行してみよう

```
#include <stdio.h>

int main()
{
   int a;
   int *pa; /* ポインタの宣言 */

   pa = &a; /* ポインタにアドレスを代入 */

   printf(“整数値を入力してください----”);
   scanf(“%d”, &a);
   printf(“変数aに%dが代入されました。\n”, a);
   printf(“変数aのアドレスは%pです。\n”, &a);
   printf(“変数aを指しているポインタはpaです。\n”);
   printf(“*paの値は%dです。\n”, *pa);

   return 0;
}
```

ポインタとは

アドレスを格納するための変数

ポインタへの値代入

```
int a;
int *pa;
```

***paをint型の変数と考えて値を代入できるか？**

- paが指し示す変数がわかっている場合は**Yes**
- 宣言しただけの場合は、paがメモリのどのアドレスを指しているか不明で変数を入れる場所がわからないので**No**

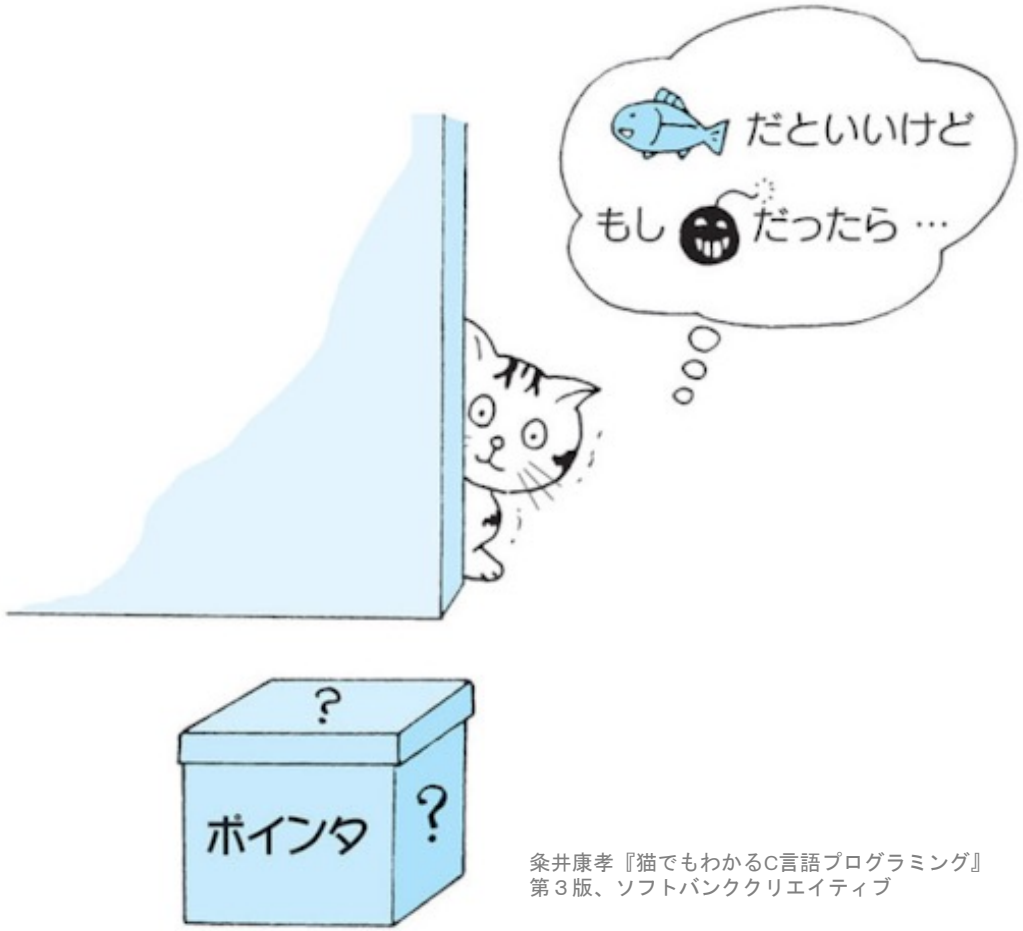
ponter02.cというファイルを作成し、右のコードを書いて実行してみよう

```
#include <stdio.h>

int main()
{
   int a;
   int *pa; /* paは不定（ゴミが入っている） */

   pa = &a; /* 変数aのアドレスを代入 */
   *pa = 5; /* aのアドレスに5を代入、つまりaに5を代入 */

   printf(“*pa = %d\n”, *pa);
   printf(“a = %d\n”, a);

   return 0;
}
```

初期化されていないポインタ

pointer02.cでpa = &a;を削除してコンパイルすると警告またはエラーが出る。実行ファイルが生成されても**絶対に実行しないこと！**

paは宣言しただけなので、中身は何が入っているかわからない。このでたらめなアドレスに5を代入すると、最悪な場合システムが壊れてしまうかもしれない。



粂井康孝『猫でもわかるC言語プログラミング』第３版、ソフトバンククリエイティブ

ポインタの宣言

- ポインタは変数なので、一度何かのアドレスを入れた後、別なアドレスに変更することも可能。
- 一般の変数と同じように宣言時にアドレスを代入して初期化できる。

ポインタの宣言

一度に複数のポインタを宣言するとき、

```
int *a, *b, *c;
```

と書くこともできる。

ポインタの宣言

```
int *a, b, c;
```

と書くと、aのみがint型へのポインタでb, cは普通のint型の変数となる。

関数の値呼び出し

右のプログラムは変数aと変数bの値を入れ替えるプログラムのはずだが、実行しても値が入れ替わらなかった。なぜか？

```
#include <stdio.h>

void swap(int, int);

int main()
{
   int a, b;

   a = 10;
   b = 20;

   swap(a, b);

   printf("a = %d, b = %d\n", a, b);

   return 0;
}

void swap(int a, int b)
{
   int c;

   c = b; /* bの値をcに代入 */
   b = a; /* aの値をbに代入 */
   a = c; /* cに入れてあった元のbの値をaに代入 */

   return;
}
```

関数の参照呼び出し

関数内で仮引数の値を変更すれば呼び出し元の実引数も変更される方法

変数aと変数bの値を入れ替えるプログラム：swap.cを書いて実行してみよう

```
#include <stdio.h>

void swap(int *, int *);

int main()
{
   int a, b;

   a = 10;
   b = 20;

  /* swap関数に変数a, bのアドレスを教える。*/
   swap(&a, &b);

   printf(“a = %d, b = %d\n”, a, b);

   return 0;
}

void swap(int *x, int *y)
{
   int z;

   z = *y;
   *y = *x;
   *x = z;

   return;
}
```

関数の参照呼び出し

変数の値を関数に変更してもらうには、変数のアドレスを渡す。

ポインタのポインタ

ポインタ自体も変数なので、メモリ内のどこかに置かれている。そのポインタ変数が格納されているアドレスを指すポインタ（**ポインタのポインタ**）を作ることができる。

ponter03.cというファイルを作成し、右のコードを書いて実行してみよう

```
#include <stdio.h>

int main()
{
   int a;
   int *p;
   int **pp;

   p = &a;
   pp = &p;

   **pp = 10;

   printf(“a = %d, *p = %d, **pp = %d\n”, a, *p, **pp);

   return 0;
}
```

**ppの意味

間接参照演算子(*)は右側の*が優先される。つまり「**pp」は、

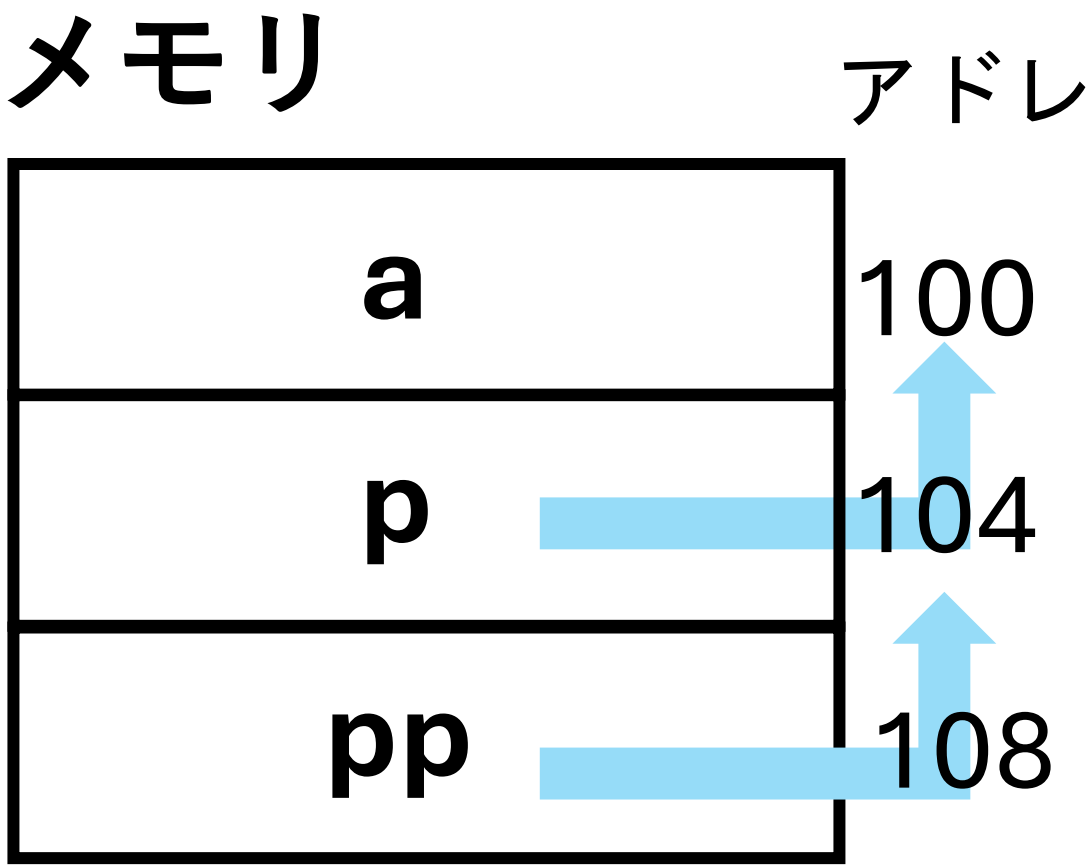
***(*pp)**

となる。「*pp」はpの値 (aのアドレス)なので、

***(p)**

と同じ。結局**ppに10を代入するとaに10が代入される。

ポインタのポインタ



「p = &a」によってppに100が代入される

「pp = &p」によってppに104が代入される

ヌルポインタ (NULL)

0番地を指すポインタ

- C言語では0番地は使わない約束になっている。
- 関数が失敗したときなどにNULLを返すものがたくさんある。
- ポインタを使うプログラムでは、使うポインタがNULLになっていないか確かめる必要がある。

今回の問い

- ポインタとは何か？
  - ポインタは変数のアドレスを格納するための変数である。
- 関数に配列を渡すにはどのようにすれば良いのか？

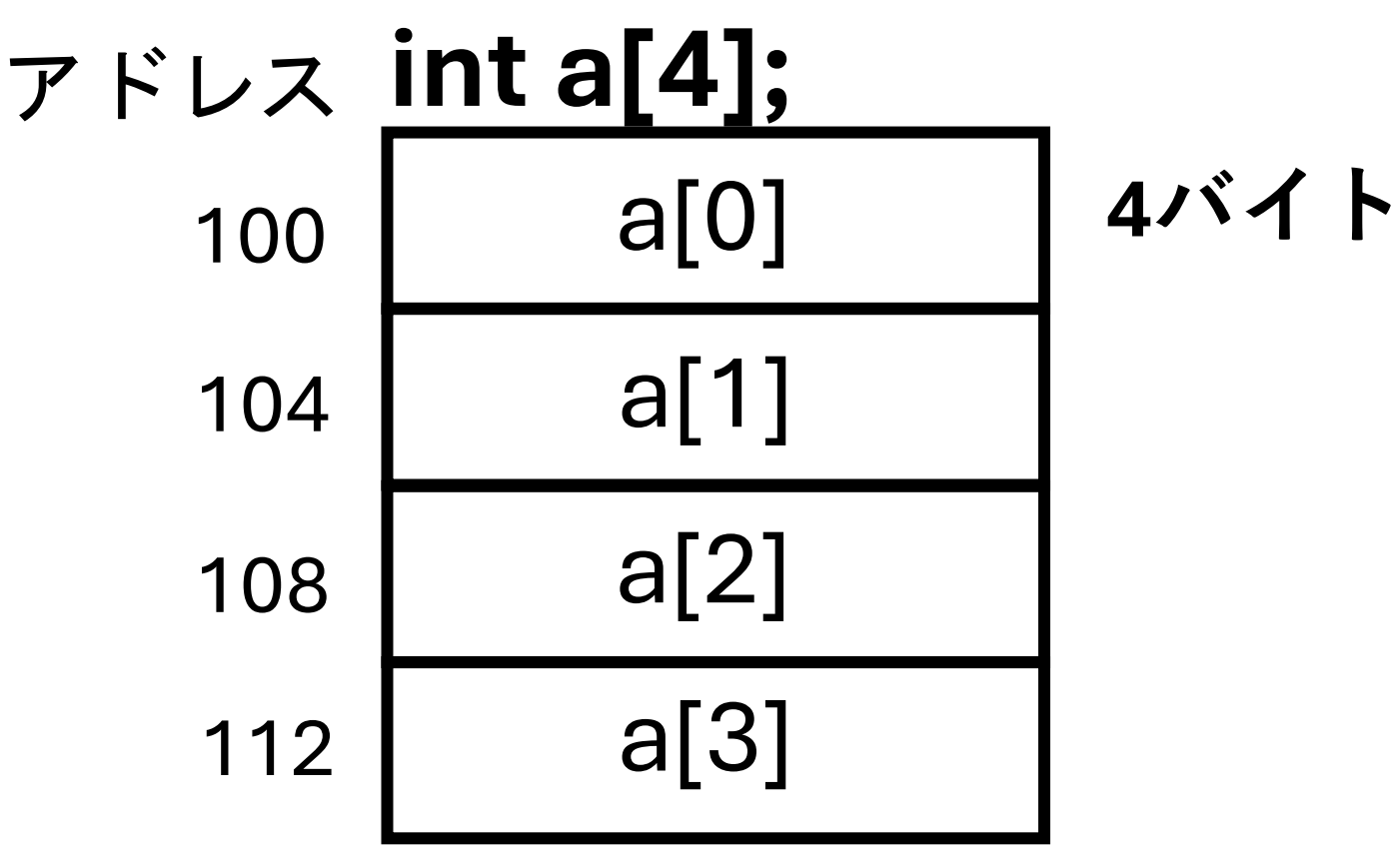
今回学ぶこと

1. メモリのアドレス
2. ポインタ
3. **配列とポインタ**
4. 配列の受け渡し

配列とアドレス

a[0], a[1], a[2], a[3]の順でメモリ上に隙間なく並んでいる

配列名は、その配列の先頭要素のアドレスに変換される



配列のメモリ内の状態

配列とアドレス

```
int a[4];
int *p;
p = a; /* pにa[0]のアドレスを代入 */
```

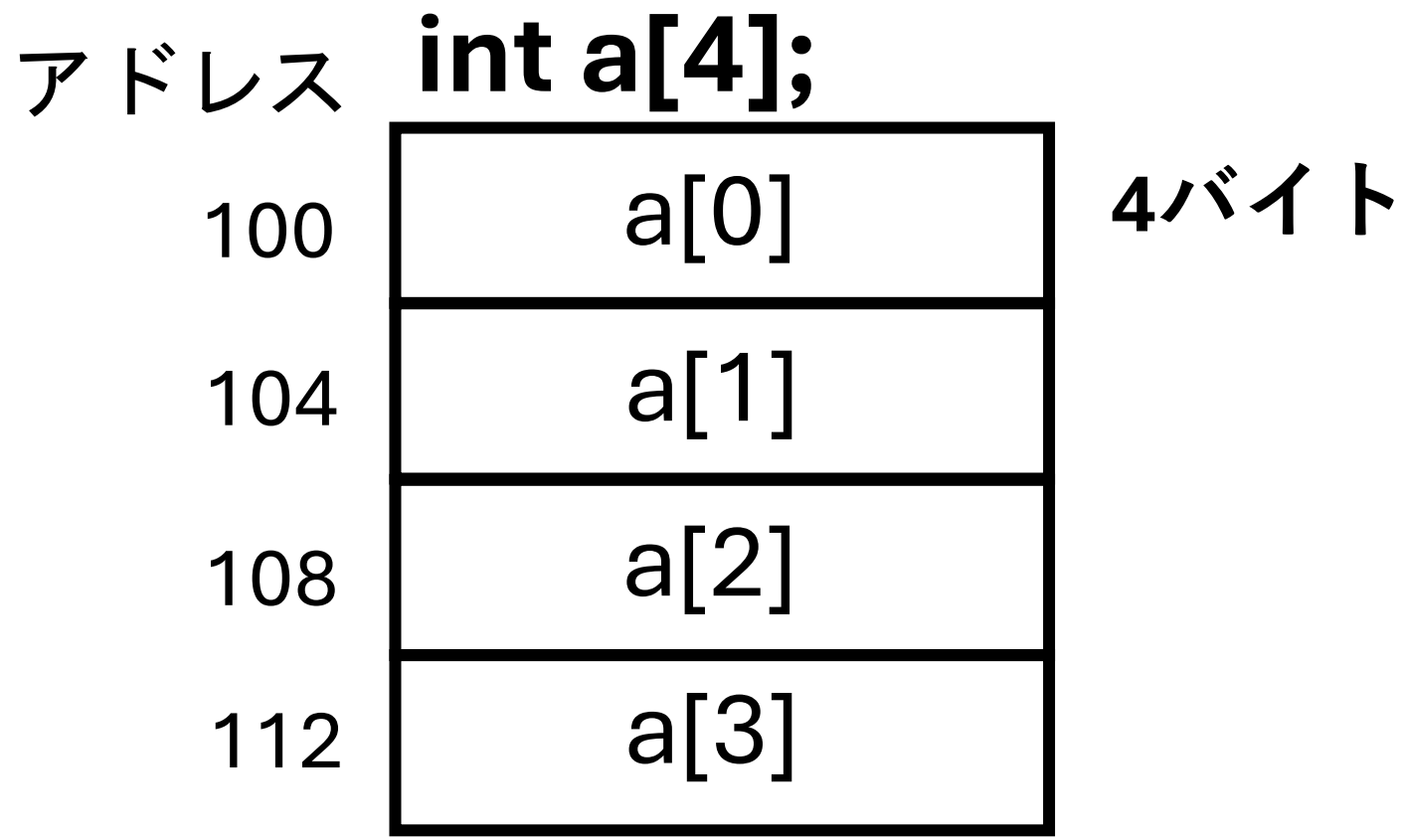
とすると、
a[0] は*pまたは*(p + 0)と同じ。
a[1] は *(p + 1)と同じ。
a[2] は *(p + 2)と同じ。
…

*(p + 1)の意味

int型へのポインタに1を加える

すなわち

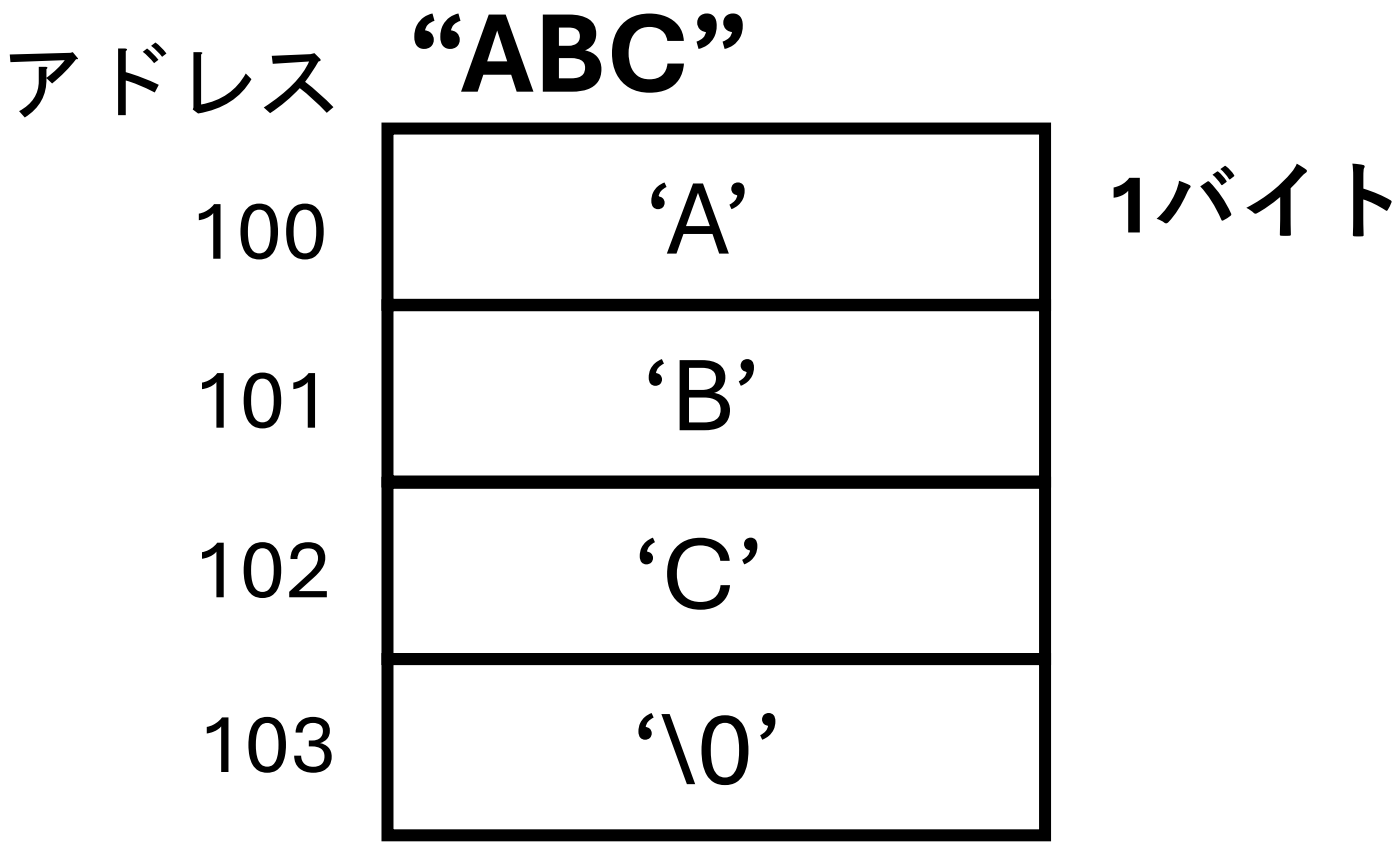
int型の大きさだけアドレスを1つ進める



配列のメモリ内の状態

文字列とアドレス

- 文字列の式の値は先頭文字の置かれたアドレス
- 文字配列最後の要素は必ず **‘\0’ (ヌル文字**)



配列のメモリ内の状態

文字列とポインタ

“ABC”は先頭文字のアドレスを表しているので、
これをchar型のポインタに代入できる。

```
char *str;
str = “ABC”;
```

または、

```
char *str = “ABC”;
```

文字列とポインタ

char *str = “ABC”;

としたとき、

- ***(str + 0) = str[0] = ‘A’**
- ***(str + 1) = str[1] = ‘B’**
- ***(str + 2) = str[2] = ‘C’**
- ***(str + 3) = str[3] = ‘\n’**

文字列ポインタと配列

次のような配列を考える

```
char *str[6];
```

これは、

```
char *(str[6]);
```

と同じ。すなわち、**char型へのポインタの配列**となっている。

文字列ポインタと配列

初期化するには、

```
char *str[6] = {
    “abc”,
    “de”,
    “fghi”,
    “jklmnop”,
    “qrs”,
    “tuvwxyz”,
};
```

とすればよい。

strings.cというファイルを作成し、右のコードを書いて実行してみよう

```
#include <stdio.h>

int main()
{
    char *str[6] = {
        “abc”,
        “de”,
        “fghi”,
        “jklmnop”,
        “qrs”,
        “tuvwxyz”,
    };
    int i;

    for(i = 0; i < 6; i++) {
        printf(“str[%d] = ‘%s’のアドレス= %p\n”, i, str[i], &str[i]);
    }

    return 0;
}
```

文字列ポインタと配列

- str[0]→文字列“abc”の先頭文字のアドレス
- str[1]→文字列“de”の先頭文字のアドレス
- str[2]→文字列“fghi”の先頭文字のアドレス
- str[3]→文字列“jklmnop”の先頭文字のアドレス
- str[4]→文字列“qrs”の先頭文字のアドレス
- str[5]→文字列“tuvwxyz”の先頭文字のアドレス

文字列ポインタと配列

- str[0][0]には‘a’
- str[0][1]には‘b’
- str[0][2]には‘c’
- str[0][3]には‘\0’

が格納されている。str[0][4], str[0][5], str[0][6], str[0][7]にはアクセスしてはいけないが、str[5][4], str[5][5], str[5][6], str[5][7]にはアクセス可能

今回学ぶこと

1. メモリのアドレス
2. ポインタ
3. 配列とポインタ
4. **配列の受け渡し**

関数への配列の受け渡し

param01.cというファイルを作成し、右のコードを書いて実行してみよう

```
#include <stdio.h>

int showarray(int *);

int main()
{
   int a[3] = {10, 20, 30};

   showarray(a);

   return 0;
}

int showarray(int x[])
{
   int i;

   for(i = 0; i < 3; i++)
      printf("x[%d] = %d\n", i, x[i]);

   return 0;
}
```

関数への配列の受け渡し

プロトタイプ宣言に

- int showarray(int x[]);
- int showarray(int x[3]);

関数定義に

- int showarray(int x[3]);
- int showarray(int *x);

と書いてもよい。

```
#include <stdio.h>

int showarray(int *);

int main()
{
   int a[3] = {10, 20, 30};

   showarray(a);

   return 0;
}

int showarray(int x[])
{
   int i;

   for(i = 0; i < 3; i++)
      printf("x[%d] = %d\n", i, x[i]);

   return 0;
}
```

多次元配列の場合

param02.cというファイルを作成し、右のコードを書いて実行してみよう

```
#include <stdio.h>

int showint(int x[][2]);

int main()
{
   int a[][2] = {
      {1, 2},
      {3, 4},
      {5, 6},
      {7, 8}
   };

   showint(a);

   return 0;
}

int showint(int m[][2])
{
   int i, j;

   for(i = 0; i < 4; i++) {
      for(j = 0; j < 2; j++) {
         printf("m[%d][%d] = %d\n", i, j, m[i][j]);
      }
   }

   return 0;
}
```

多次元配列の場合

showint関数は、

- int showint(int **)
- int showint(int (*m)[2])

とも書ける。

```
#include <stdio.h>

int showint(int x[][2]);

int main()
{
   int a[][2] = {
      {1, 2},
      {3, 4},
      {5, 6},
      {7, 8}
   };

   showint(a);

   return 0;
}

int showint(int m[][2])
{
   int i, j;

   for(i = 0; i < 4; i++) {
      for(j = 0; j < 2; j++) {
         printf("m[%d][%d] = %d\n", i, j, m[i][j]);
      }
   }

   return 0;
}
```

今回の問い

- ポインタとは何か？
  - ポインタは変数のアドレスを格納するための変数である。
- **関数に配列を渡すにはどのようにすれば良いのか？**
  - **関数の引数に配列を指定する。**

もっと学びたい方へ


新・標準
プログラマーズ
ライブラリ
C言語
ポインタ
完全制覇
前橋和弥
Kazuya Maebashi
Cはなんでこんな言語に「なっちゃった」のか。そもそもこの悪名高いポインタとは何か。
初心者が必ずひっかかる、配列とポインタのまぎらわしい文法とは。
Cはメモリを実際にどんなふうに使うのか。Cの宣言は英語で読め。
ポインタの真の使い方は。Cの文法を深く知ることで見えてくること納得できること。
技術評論社

今回学んだこと

1. メモリのアドレス
2. ポインタ
3. 配列とポインタ
4. 配列の受け渡し

今回の目標

「ポインタ」の概念を理解し、
使いこなせるようになる。

課題

第2回課題の名前を入れる配列をポインタを用いて定義せよ。それを用いて第2回課題と同様に３名の名前と英語、数学の点数、各教科の平均点を表示するプログラムを作成せよ。生成AIでのコード生成は不可。

提出先：yutaka.hirai@koeki-u.ac.jp

件名：応用プログラミング第6回課題 [学籍番号]

締め切り：11月12日 (木)

次回

第１回　C言語の基本文法・言語処理系
第２回　数学的計算、配列、文字列
第３回　型変換、入出力処理
第４回　デバッグ、関数定義、変数の分離
第５回　プロトタイプ宣言、分割コンパイル
第６回　配列の受け渡し、ポインタ
第７回　まとめ、期末試験