

#07

## ポイントの基礎

2022 年度 / プログラミング及び実習 III

角川裕次

龍谷大学 先端理工学部

# もくじ

## 1 第 10-1 節 ポインタ

## 2 第 10-2 節 ポインタと関数

## 今回 (#07) の内容：シラバスでの該当部分

小テーマ：ポインタの基礎

第 10 回：アドレス演算子とポインタ

第 11 回：ポインタを利用したプログラミング

# 重要概念リスト

- ポインタ
- アドレス演算子 &
- 間接演算子 \*
- 空ポインタ NULL

## 今回の実習・課題 (manaba へ提出)

実習内容と課題内容は講義途中に提示します

(作成したファイル類は manaba に提出)

## 第 10-1 節 ポインタ

Q : 出力はどうなる?

- 整数 A に 57, 整数 B に 21 を入力

List 10-1 : 2 つの整数の和と差を求める関数 (間違い)

```
#include <stdio.h>
void sum_diff(int n1, int n2, int sum, int diff)
{
    sum  = n1 + n2;    /* 和 */
    diff = (n1 > n2) ? n1 - n2 : n2 - n1; /* 差 */
}
int main(void)
{
    int na, nb;
    int wa = 0, sa = 0;
    puts("二つの整数を入力せよ。");
    printf("整数A :"); scanf("%d", &na);
    printf("整数B :"); scanf("%d", &nb);
    sum_diff(na, nb, wa, sa);
    printf("和は%dで差は%dです。 \n", wa, sa);
    return 0;
}
```

## 【実行してみた】

二つの整数を入力せよ。

整数 A : 57

整数 B : 21

和は0で差は0です。

■  $57 + 21 = 78$

■  $57 - 21 = 36$

期待したのと違う

「和は78で差は36です。」と出て欲しかったのに



## 関数の引数：値渡し

関数内で書き換えた仮引数の値：呼び出し元には戻らない

```
void sum_diff(int n1, int n2, int sum, int diff)
{
    sum = n1 + n2;    /* 和 */
    diff = (n1 > n2) ? n1 - n2 : n2 - n1; /* 差 */
}
```

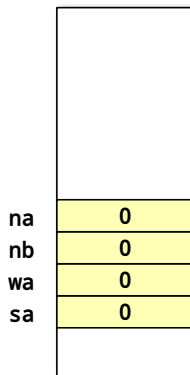
関数内で仮引数の sum と diff に代入している

# 実行過程に於けるスタックの変化 (1)

main 関数での変数宣言 (スタック上に na, nb, wa, sa の 4 変数)

```
void sum_diff( int n1, int n2,  
               int sum, int diff ) {  
    sum = n1 + n2;  
    diff = n1 - n2;  
}  
  
→ int main(void){  
    int na, nb;  
    int wa = 0; sa = 0;  
    scanf("%d", &na);  
    scanf("%d", &nb);  
    sum_diff(na, nb, wa, sa);  
    printf("和=%d 差=%d\n", wa, sa);  
    return 0;  
}
```

ソースコード



スタック

## 実行過程に於けるスタックの変化 (2)

scanf で値を読み込む (57 と 21)

```
void sum_diff( int n1, int n2,  
               int sum, int diff ) {  
    sum = n1 + n2;  
    diff = n1 - n2;  
}  
  
int main(void){  
    int na, nb;  
    int wa = 0; sa = 0;  
    scanf("%d", &na);  
    scanf("%d", &nb);  
    sum_diff(na, nb, wa, sa);  
    printf("和=%d 差=%d\n", wa, sa);  
    return 0;  
}
```

ソースコード

na	57
nb	21
wa	0
sa	0

スタック

## 実行過程に於けるスタックの変化 (3)

sum\_diff 関数の呼出 (スタック上に n1, n2, sum, diff の 4 仮引数)

```
void sum_diff( int n1, int n2,  
               int sum, int diff ) {  
    sum = n1 + n2;  
    diff = n1 - n2;  
}  
  
int main(void){  
    int na, nb;  
    int wa = 0; sa = 0;  
    scanf("%d", &na);  
    scanf("%d", &nb);  
    sum_diff(na, nb, wa, sa);  
    printf("和=%d 差=%d\n", wa, sa);  
    return 0;  
}
```

ソースコード


n1	57
n2	21
sum	0
diff	0
na	57
nb	21
wa	0
sa	0

スタック

## 実行過程に於けるスタックの変化 (4)

sum\_diff 関数の本体の実行 (sum と diff への代入)

main 関数での変数 wa と sa へは代入されない (値は 0 のまま)



```
void sum_diff( int n1, int n2,  
               int sum, int diff ) {  
    sum = n1 + n2;  
    diff = n1 - n2;  
}  
  
int main(void){  
    int na, nb;  
    int wa = 0; sa = 0;  
    scanf("%d", &na);  
    scanf("%d", &nb);  
    sum_diff(na, nb, wa, sa);  
    printf("和=%d 差=%d\n", wa, sa);  
    return 0;  
}
```

ソースコード

n1	57
n2	21
sum	78
diff	36
na	57
nb	21
wa	0
sa	0

スタック

## 実行過程に於けるスタックの変化 (5)

sum\_diff の終了時にはスタック上の 4 仮引数を開放  
main 関数で wa と sa を表示 (値 0 を表示)

```
void sum_diff( int n1, int n2,  
               int sum, int diff ) {  
    sum = n1 + n2;  
    diff = n1 - n2;  
}  
  
int main(void){  
    int na, nb;  
    int wa = 0; sa = 0;  
    scanf("%d", &na);  
    scanf("%d", &nb);  
    sum_diff(na, nb, wa, sa);  
    printf("和=%d 差=%d\n", wa, sa);  
    return 0;  
}
```

ソースコード

na	57
nb	21
wa	0
sa	0


スタック

# 関数内での変更を呼び出し元に反映したいです

Q: sum\_diff の引数を wa と sa に変更すれば OK ですか?

A: だめです. 結果は変わらないです.

同じ名前でも記憶場所が違うから別物のままです.



```
void sum_diff( int n1, int n2,  
              int wa, int sa ) {  
    wa = n1 + n2;  
    sa = n1 - n2;  
}
```

```
int main(void){  
    int na, nb;  
    int wa = 0; sa = 0;  
    scanf("%d", &na);  
    scanf("%d", &nb);  
    sum_diff(na, nb, wa, sa);  
    printf("和=%d 差=%d\n", wa, sa);  
    return 0;  
}
```

ソースコード

n1	57
n2	21
wa	78
sa	36
na	57
nb	21
wa	0
sa	0

スタック

# オブジェクトとアドレス p.277

Q. 関数内での変更を呼び出し元に反映するにはどうするか?

A. **ポインタ**を使用すれば可能

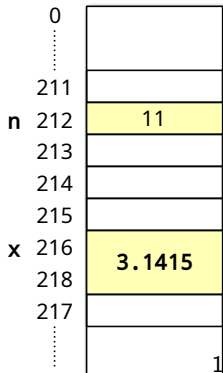
- **オブジェクト (変数の実体) の配置アドレス**を関数に渡す
- 関数内では指定されたアドレスの内容を更新する
- 呼び出し元には更新結果が反映される

int n;

11

double x;

3.1415



**ポインタ (pointer) : メモリアドレスの抽象化**



## アドレス演算子 & p.278

### アドレス演算子 &

オブジェクト (変数等) が配置されているアドレスを得る演算子

- オブジェクトを指すポインタを得る

例 : &a

例 : &c[20]

例 : &m[10][1]

# アドレス演算子の使用例

List 10-2 : オブジェクトのアドレスを表示

```
#include <stdio.h>
int main(void)
{
    int    n;
    double x;
    int    a[3];
    printf("n    の ア ド レ ス : %p\n", &n);
    printf("x    の ア ド レ ス : %p\n", &x);
    printf("a[0] の ア ド レ ス : %p\n", &a[0]);
    printf("a[1] の ア ド レ ス : %p\n", &a[1]);
    printf("a[2] の ア ド レ ス : %p\n", &a[2]);
    return 0;
}
```

**& : アドレス演算子 (オブジェクトのアドレスを得る演算子)**

- printf での変換指定 %p : ポインタの表示を指示

# 実行結果の例

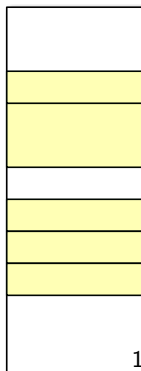
```
n    の ア ド レ ス : 0 x7fffcc0be6fc  
x    の ア ド レ ス : 0 x7fffcc0be700  
a[0] の ア ド レ ス : 0 x7fffcc0be70c  
a[1] の ア ド レ ス : 0 x7fffcc0be710  
a[2] の ア ド レ ス : 0 x7fffcc0be714
```

(実行環境により表示内容は異なる; x86\_64 Ubuntu ビットにて実行)

```
int main(void){  
    int n;  
    double x;  
    int a[3];  
  
    ...  
  
    return 0;  
}
```

ソースコード

```
n  0x7fffcc0be6fc  
   0x7fffcc0be700  
x  0x7fffcc0be704  
   0x7fffcc0be708  
a[0] 0x7fffcc0be70c  
a[1] 0x7fffcc0be710  
a[2] 0x7fffcc0be714
```



List 10-3 (p の値: n へのポインタ値)

```
#include <stdio.h>
int main(void)
{
    int n = 57;
    printf("n = %d\n", n);
    printf("&n = %p\n", &n);
    int *p = &n;
    printf("p = %d\n", p);
    printf("*p = %p\n", &p);
    return 0;
}
```

n 0x7fffe257ea8c

p 0x7fffe257ea90

57

0x7fffe257ea8c

実行結果の例 (p の値: n の配置アドレス)

```
n = 57
&n = 0x7fffe257ea8c
p = 0x7fffe257ea8c
*p = 57
```

# ポインタ型

型  $T^*$  :  $T$  型オブジェクトを指すポインタ型

- $\text{int}^*$  :  $\text{int}$  型オブジェクトを指すポインタ型

ポインタが指すオブジェクトの型によってポインタ型も異なる

ポインタ型	説明
$\text{int}^*$	$\text{int}$ 型オブジェクトを指すポインタ
$\text{double}^*$	$\text{double}$ 型オブジェクトを指すポインタ
$\text{char}^*$	$\text{char}$ 型オブジェクトを指すポインタ

ポインタを指すポインタあり

ポインタ型	説明
$\text{char}^{**}$	$\text{char}^*$ 型オブジェクトを指すポインタ
$\text{int}^{**}$	$\text{int}^*$ 型オブジェクトを指すポインタ

# ポインタ変数の宣言の例

## 宣言の例

```
int *p;  
int  a, b, *q, i;  
char ch, c, *s, *t;  
double z;  
double *xp, *yp, th;
```

## 上記各変数の型

- int 型の変数 : a, b, i
- int\*型の変数 (ポインタ) : p, q
- char 型の変数 : ch, c
- char\*型の変数 (ポインタ) : s, t
- double 型の変数 : z, th
- double\*型の変数 (ポインタ) : xp, yp

## 失敗例あるある

がくせい: 「int\* 型の変数 p と q を宣言しますねー」

```
int*  p, q;
```

せんせい: 「q は int 型になってますよ」

## 間接演算子 \* p.281

## 間接演算子 \*

\*p : p が指すオブジェクトを表す

■ p : ポインタ変数

List 10-4 (部分) 999 の代入先が x か y かを sw の値で選択

```
int x = 124;
int y = 456;
... 略 (sw に値を設定) ...
int *p;
if (sw == 0) {
    p = &x;
} else {
    p = &y;
}
*p = 999; // sw が 0 のときは x へ , 1 のときは y へ 999 を代入
printf("x=%d\n", x);
printf("y=%d\n", y);
return 0;
}
```



## 間接演算子: 代入文の左辺と右辺

代入文の左辺の場合:

右辺の値をポインタが指す先に書き込む

```
int *p;  
*p = ....;
```

代入文の右辺の場合:

ポインタが指しているオブジェクトの値を表す

```
int *p, x;  
x = ... + *p + ....;
```

# ポインタ変数の宣言と間接演算子の違いに注意

変数宣言 (初期化あり) の例 :

ポインタ変数 p を宣言

```
int x;  
int *p = &x;
```

間違い理解: p が指しているオブジェクトに &x の値を書き込んで初期化

正しい理解: p の初期値を x を指すポインタ値に設定

以下と同じだと思きましょう

```
int x;  
int *p;  
p = &x;
```

## 間接演算子の使用上の注意

\*p = ...; の実行時に p はオブジェクトを指していること

もしそうでないと発生しうる困ること

- アクセス可能なメモリ範囲外へのアクセス発生
- 値が変わっては困る変数の値の変化

駄目プログラム例：p が指すオブジェクトが不定

```
int main(void)
{
    int x = 0;
    int *p;
    *p = *p + 1;    // pは何を指している ???
    printf("x=%d\n", x);
    return 0;
}
```

変なメモリアドレスの内容を書き換えようとして自滅する場合あり

Segmentation fault (core dumped)

(そうならない場合もある; だからといって正しいプログラムではない)

## 第 10-2 節 ポインタと関数

## 関数の引数としてのポインタ p.284

int 型変数に 999 を代入する関数を作りたい

- 対象の変数は関数の引数で与えられる

List 10-5 (部分) ポインタを使ってこのように書く

```
void set999(int *p)
{
    *p = 999;    // 指定のメモリアドレスに書き込む
}

int main(void) {    // 呼び出し例
    int x;
    set999(&x);    // メモリアドレス(ポインタ)を渡す
                  (xの値は999に変化) ...
}
```

失敗例：p への変更は呼び出し元には戻らない

```
void set999(int p)
{
    p = 999;
}
```

## 和と差を求める関数 p.286

List 10-6 (部分) : 和と差を求めてポインタにより呼び出し元に結果を反映

```
void sum_diff(int n1, int n2, int *sum, int *diff)
{
    *sum  = n1 + n2;
    *diff = (n1 > n2) ? n1 - n2 : n2 - n1;
}

int main(void)
{
    int na, nb;
    int wa = 0, sa = 0;
    ... 略 ...
    sum_diff(na, nb, &wa, &sa);
    ... 略 ...
}
```

&wa : 変数 wa を指すポインタ

- この値 (ポインタ値) が値渡しで関数へ渡される
- &sa も同様

# 和と差を求める; main 関数

List 10-6 (部分) : main 関数部

```
int main(void)
{
    int na, nb;
    int wa = 0, sa = 0;
    puts("二つの整数を入力せよ。");
    printf("整数A:");    scanf("%d", &na);
    printf("整数B:");    scanf("%d", &nb);
    sum_diff(na, nb, &wa, &sa);
    printf("和は%dで差は%dです。 \n", wa, sa);
    return 0;
}
```

注目ポイント：変数 wa と sa の値が変更されている

実行例

```
二つの整数を入力せよ。
整数 A : 57
整数 B : 21
和は78で差は36です。
```

## 2 値の交換 p.287

List 10-7 (部分) : main 関数部

```
int main(void)
{
    int na, nb;
    puts("二つの整数を入力せよ。");
    printf("整数A : ");    scanf("%d", &na);
    printf("整数B : ");    scanf("%d", &nb);
    swap(&na, &nb);
    puts("これらの値を交換しました。");
    printf("整数Aは%dです。 \n", na);
    printf("整数Bは%dです。 \n", nb);
    return 0;
}
```

実行例

```
二つの整数を入力せよ。
整数 A : 57
整数 B : 21
これらの値を交換しました。
整数 A は21です。
整数 B は57です。
```



## 2つの値を交換する関数

List 10-6 (部分) : 二つの整数値を交換する

```
void swap(int *px, int *py)
{
    int temp = *px;
    *px = *py;
    *py = temp;
}
int main(void)
{
    int na, nb;
    ... 略 ...
    swap(&na, &nb);
    ... 略 ...
}
```

関数 swap : ポインタを使うことで2つの値の交換を実現

- 呼び出し元の変数の値を交換

変数 na と nb の値を昇順にソート

List 10-8 (部分) : 2 つの整数を昇順に並べる

```
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

void sort2(int *n1, int *n2) /* *n1  *n2 に並べる */
{
    if (*n1 > *n2)
        swap(n1, n2);
}

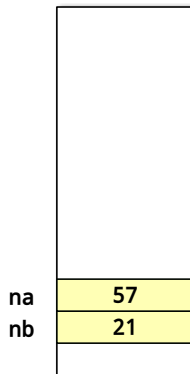
int main(void)
{
    int na, nb;
    ... 略 ...
    sort2(&na, &nb);
    ... 略 ...
}
```

# 実行の過程 (1/3)

main 関数に入った時 (変数に値を設定した後)

```
void swap( int *x, int *y )  
{  
    int temp = *x;  
    *x = *y;  
    *y = temp;  
}  
void sort2( int *n1, int *n2 )  
{  
    if (*n1 > *n2)  
        swap(n1, n2);  
}  
int main(void)  
{  
    int na, nb;  
    ... 略 ...  
    sort2(&na, &nb);  
    ... 略 ...  
}
```

ソースコード



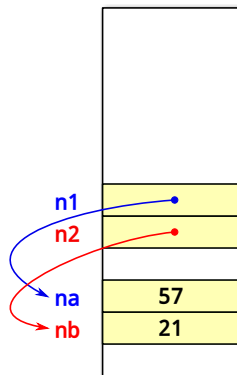
スタック

## 実行の過程 (2/3)

sort2 関数に入った時

```
void swap( int *x, int *y )  
{  
    int temp = *x;  
    *x = *y;  
    *y = temp;  
}  
void sort2( int *n1, int *n2 )  
{  
    if (*n1 > *n2)  
        swap(n1, n2);  
}  
int main(void)  
{  
    int na, nb;  
    ... 略 ...  
    sort2(&na, &nb);  
    ... 略 ...  
}
```

ソースコード



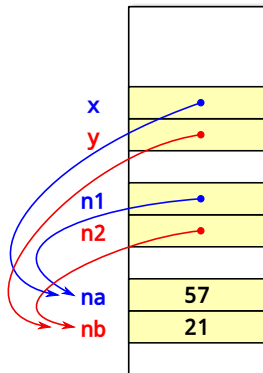
スタック

## 実行の過程 (3/3)

swap 関数に入った時 : \*px と \*py の値の交換は na と nb の値の交換

```
void swap( int *x, int *y )  
{  
    int temp = *x;  
    *x = *y;  
    *y = temp;  
}  
void sort2( int *n1, int *n2 )  
{  
    if (*n1 > *n2)  
        swap(n1, n2);  
}  
int main(void)  
{  
    int na, nb;  
    ... 略 ...  
    sort2(&na, &nb);  
    ... 略 ...  
}
```

ソースコード



スタック

## swap 呼び出し: 間違いあるある

List 10-8 を以下のようにしてはダメ

```
void swap(int *px, int *py)
{
    ...
}
void sort2(int *n1, int *n2)
{
    ...
    if (*n1 > *n2)
        swap(&n1, &n2); /* ダメ (=ポインタへのポインタ) */
    ...
}
```

swap の各引数の型: int\* (int へのポインタ)

n1 と n2 の型: int\* (int へのポインタ)

&n1 と &n2 の型: int\*\* (int へのポインタのポインタ)

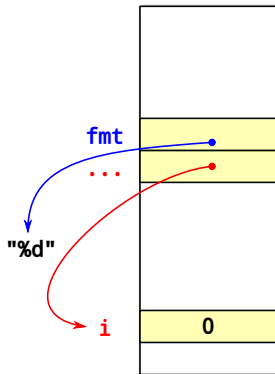
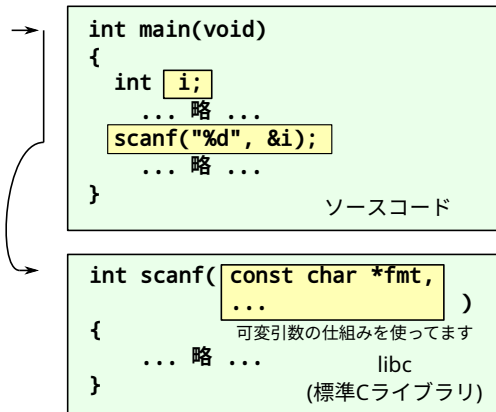
- 呼び出し swap(&n1, &n2) は型の不一致でエラー
- 呼び出し swap(n1, n2) が正しい

# scanf 関数とポインタ

p.290

```
scanf("%d", &i);
```

- scanf 関数に整数変数へのポインタを渡す
- (scanf といえども普通の関数; ポインタを使うしかない)



# 空ポインタ

p.290

## 空ポインタ NULL

オブジェクトを指していない特別なポインタ (定数値)

ヘッダ `stddef.h` でマクロ定義

以下のいずれのヘッダのインクルードでもよい

- `stdio.h`, `stddef.h`
- `stdlib.h`, `string.h`, `time.h`

注意: `NULL` は 0 と定義されていることが多いが処理系により異なる

- ソースコードでは記号名 `NULL` を使うこと
- 0 は使わない
- ソースコードの可搬性のため



# スカラ型 p.291

スカラ型 (scalar type) : 演算可能な値を持つデータ型  
ポインタはスカラ型

int						
short int						
long int	符号付き 整数型					
long long int						
signed char						
unsigned int		整数型				
unsigned short int	符号無し 整数型					
unsigned long int			実数型	算術型	スカラ型	オブジェ クト型
unsigned long long int						
unsigned char						
char	文字型					
enum	列挙型					
float						
double	浮動小数点型					
long double						
T*	ポインタ型					
T[ ]	配列型					
struct	構造体型				集成体型	
union	共用体型					

# ポインタの型

メモリアドレスはただの数値

- 指している先のデータ型が何かわからないのでバグの元

C 言語での解決法: ポインタに型情報を持たせる

- C コンパイラはコンパイル時にデータ型をチェック
- 型間違いを検出して型間違いのバグがないようにする

List 10-8 : 型の不一致

```
void swap(int *px, int *py)
{
    ... 略 ...
}

int main(void)
{
    double da, db;
    ... 略 ...
    swap(&da, &db);
    ... 略 ...
}
```

# ポインタの型を間違えるとコンパイル時に警告される

## コンパイル時に警告エラー

```
$ cc      list1008.c  -o list1008
list1008.c: In function 'main':
list1008.c:23:7: warning: passing argument 1 of 'swap' from
incompatible pointer type [-Wincompatible-pointer-types]
    swap(&da, &db);
        ^
list1008.c:8:6: note: expected 'int *' but argument is of
type 'double *'
    void swap(int *px, int *py)
           ^~~~
```

超訳：「型が変だが本当にそれでいいのか？」

- swap の第 1 引数の型を double\* で呼び出してますね
- &da のところなんですが
- swap のところでは型は int\* と書いてましたよ

警告メッセージがなくなるまで修正すべし

- 実行はできるかもしれないが変な結果になる

ポインタの型の意義：型の間違いの検出

- 指すオブジェクトの型によりポインタの型を決めて区別

おわり