情報工学演習 1
第 8 回

分割コンパイル，深いコピー，浅いコピー

今日の内容

- 分割コンパイル
- 深いコピー，浅いコピー

分割コンパイル

2024/6/6

分割コンパイル

- プログラムをすべて同じファイルに記述するのではなく，複数のプログラムに分けて記述し，コンパイルすること
- 分割コンパイルが必要な理由
  - １つのプログラムが長くなりすぎるのを防ぐ
  - 複数人でプログラムを扱うときの利便性の向上
  - 過去のプログラムを再利用するのに便利

復習：関数のプロトタイプ宣言

- 関数の宣言を先に書いて，定義を後に書く

main.c

プロトタイプ宣言

```
#include <stdio.h>

int wa( int a, int b );

int main (void){
   int a, b, sum;
   a=3;
   b=4;
   sum=0;
   sum = wa( a, b );
   printf( "%d", sum);
   return 0;
}
```

```
int wa( int a, int b ){
    int z;
    z = a+b;
    return z;
}
```

関数の定義

関数の定義のみを記すファイル

main.c

```
#include <stdio.h>

int wa( int a, int b );

int main (void){
   int a, b, sum;
   a=3;
   b=4;
   sum=0;
   sum = wa( a, b );
   printf( "%d", sum);
   return 0;
}
```

wa.c

```
int wa( int a, int b ){
    int z;
    z = a+b;
    return z;
}
```

コンパイルの方法

- 各.c ファイルをコンパイルする

  gcc -c main.c → main.o の生成

  gcc -c wa.c → wa.o の生成

- 統合することでmain 関数が動く

  gcc -o main main.o wa.o → main の生成

関数の宣言を記すファイル（ヘッダファイル）

main.c

```
#include <stdio.h>

#include “wa.h”

int main (void){
   int a, b, sum;
   a=3;
   b=4;
   sum=0;
   sum = wa( a, b );
   printf( “%d”, sum);
   return 0;
}
```

wa.c

```
int wa( int a, int b ){
    int z;
    z = a+b;
    return z;
}
```

wa. h

```
int wa( int a, int b );
```

スライドp.6 のコマンドで
同様に分割コンパイル可能

深いコピー，浅いコピー

2024/6/6

ポインタ復習サンプルコードその1

```c
#include <stdio.h>

int main(void) {
    char mychar = 55;
    int mydt = 1234;

    printf( "mychar is %d.¥n", (int)mychar );
    printf( "The address of mychar is %p.¥n", &mychar );

    printf( "mydt is %d.¥n", mydt);
    printf( "The address of mydt is %p.¥n", &mydt);

    return 0;
}
```

数値の出力

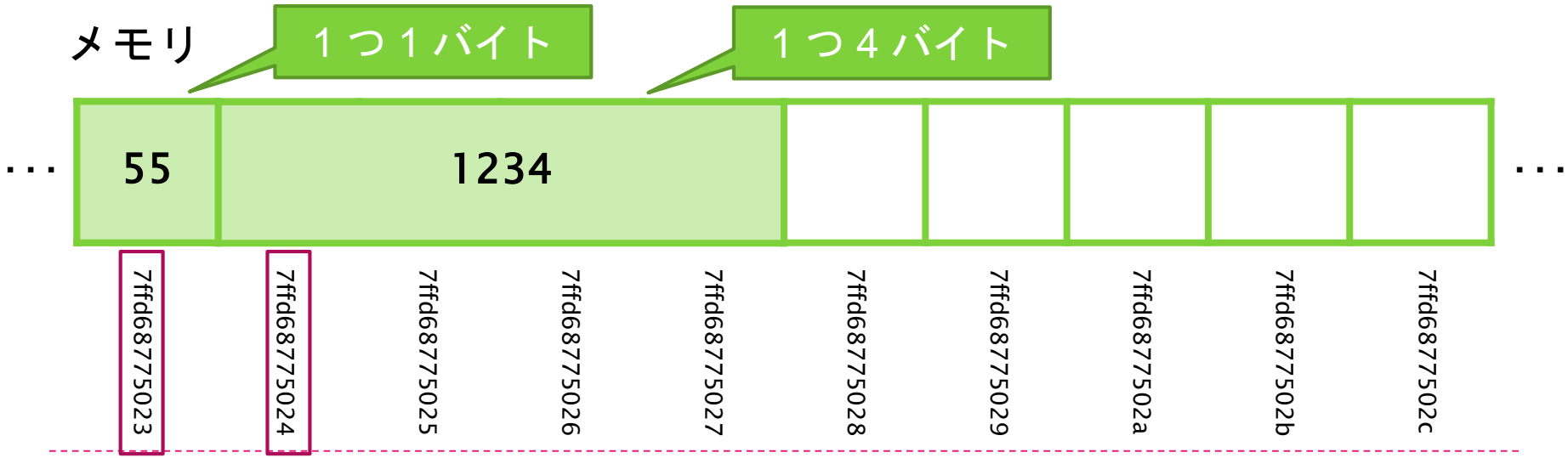
アドレスの出力

```
mychar is 55.
The address of mychar is 0x7ffd68775023.
mydt is 1234.
The address of mydt is 0x7ffd68775024.
```

アドレス

アドレスとは

- 計算機内のメモリに０から割り振られた番号のこと
  - １バイトごとに１つ割り当て
  - 16 進数で表現
  - 計算機環境によりアドレスの桁数は異なる
    - repl.it では 8 バイト（16 桁）
  - `&` 演算子でアクセス可能（変数 `a` のアドレス: `&a`）



ポインタとは？

- メモリ領域上の**アドレスを格納する変数**

ポインタ型の変数の宣言
変数型名* 変数名;
例：`char* mypt;`

- 変数型毎にポインタの型が存在
  - どの変数型のポインタ型でもバイト数は同じ
    - アドレスのバイト数が同じなため
  - 異なる変数型のポインタは型が異なると認識

```
char* pt_c;
float* pt_f;
```

✖ `pt_c = pt_f`　　⭕ `pt_c = (char* )pt_f`

- ポインタから値を参照可能（実態）
  - * 演算子の利用(変数 a の実態：*a)

ポインタ復習サンプルコードその2

```c
#include <stdio.h>

int main(void) {
    char mychar = 55;
    int mydt = 1234;

    char *pt_c;
    int *pt_i;

    pt_c = &mychar;
    pt_i = &mydt;

    printf( "pt_c: %p, %d¥n", pt_c, (int)*pt_c );
    printf( "pt_i: %p, %d¥n", pt_i, *pt_i );

    return 0;
}
```

実行結果

```
pt_c: 0x7ffc6f12c233, 55
pt_i: 0x7ffc6f12c234, 1234
```

ポインタを用いた配列の表現

- 配列の長さは宣言時に決定
  - 例：`int a[10];`
    `int b[]={2, 4, 6, 8}; //` 長さは `4`

- 配列の長さを処理に応じて決定したい
  
  ポインタと動的なメモリ確保による配列表現

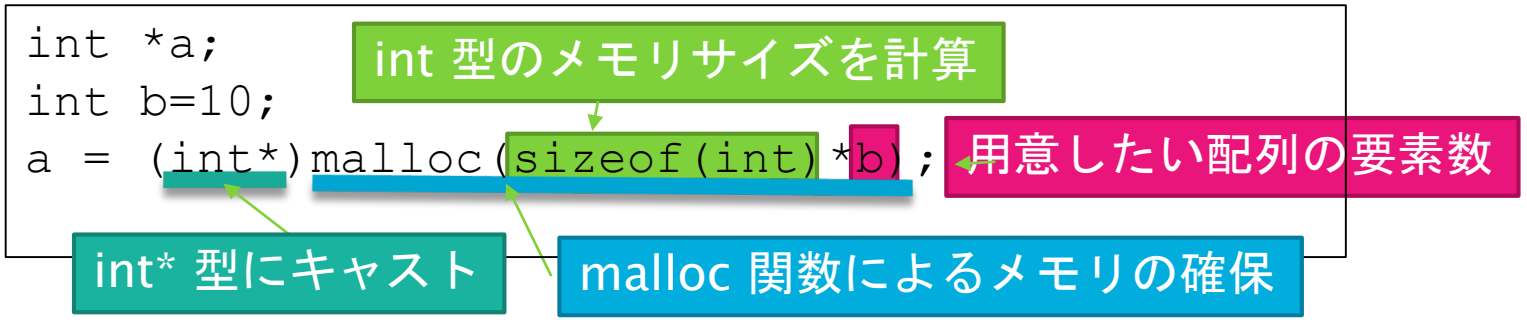
動的メモリ確保による配列の表現
サンプルコード

```
#include <stdio.h>
#include <stdlib.h>
int main( void ){
  int *a;
  int b=10, i;
  a = (int*)malloc(sizeof(int)*b);
  if( a == NULL ){
    printf( "cannot obtain memories.¥n" );
    exit( -1 );
  }
  for( i=0; i<b; i++ ){
    a[i] = i;
  }
  for( i=0; i<b; i++ ){
    printf( "%d¥n", a[i] );
  }
  free( a );
  return 0;
}
```

実行結果

```
0
1
2
3
4
5
6
7
8
9
```

malloc 関数と sizeof 関数



- malloc 関数（指定バイト分，メモリを確保）
  - 書式：void* malloc(size_t size);
  - 引数：size_t size 確保したメモリのバイトサイズ
  - 戻り値：成功時は確保したメモリブロックを指すポインタ．失敗時は NULL
- sizeof 演算子（型や変数のメモリサイズ（バイト数）を返す）
  - 書式：sizeof(型)，sizeof(変数)，sizeof(配列)

コピーの方法に種類がある？

- ポインタのコピーをよく理解していないと，こちらが予期せぬ値の変更が起きる可能性がある
- コピーの種類
  - 浅いコピー
    - ポインタのアドレスのみをコピーする

コピー元とコピー先が同じデータを指す

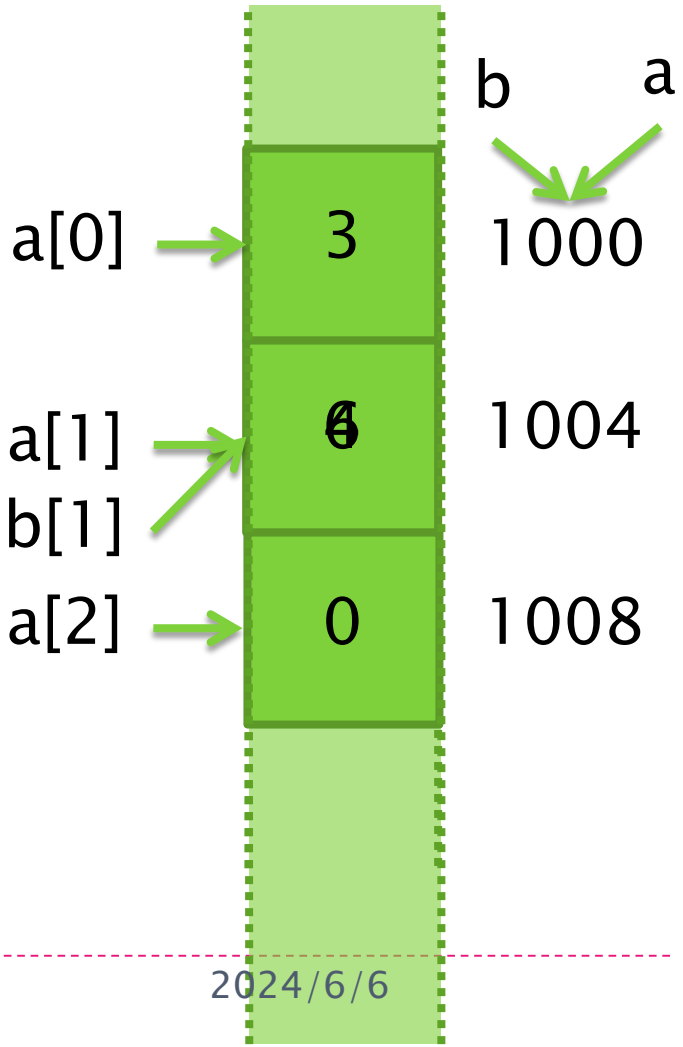
  - 深いコピー
    - ポインタの実態のデータをコピー先で確保したメモリ上に代入

コピー元とコピー先が別のデータを指す

浅いコピーのメモリ空間上での挙動

```
int main (void){

   int *a, *b;

a=(int*)malloc(sizeof(int)*3);
   a[0] = 3;
   a[1] = 4;
   a[2] = 0;

   b=a;
   b[1]=6;

   printf( "%d", a[1] );
   return 0;
}
```

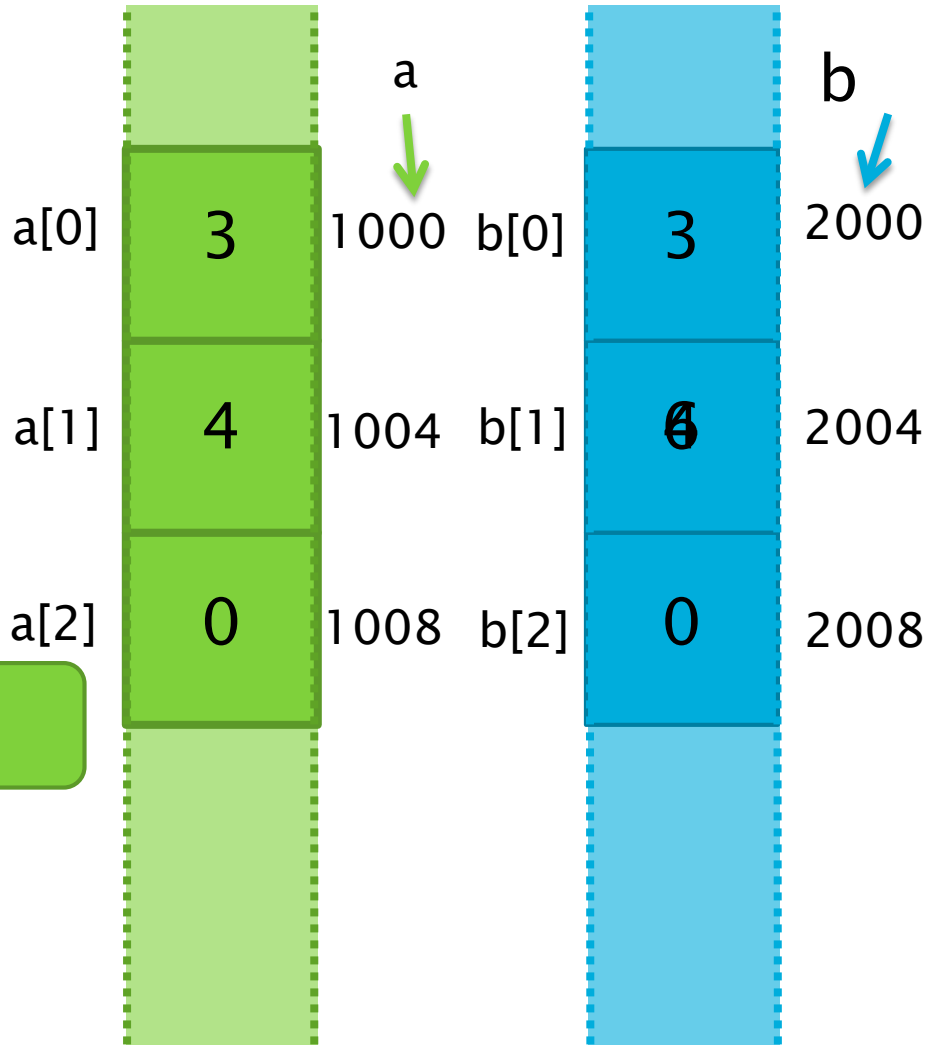
6



深いコピーのメモリ空間上での挙動

```
int main (void){

  int *a, *b;
  int i;

a=(int*)malloc(sizeof(int)*3);
  a[0] = 3;
  a[1] = 4;
  a[2] = 0;

b=(int*)malloc(sizeof(int)*3);
  for( i=0; i<3; i++ ){
      b[i]=a[i];
  }
  b[1]=6;
  printf( "%d", a[1] );
  return 0;
}
```

4



関数と引数

main 関数

```
int main (void){

   int a, b, sum;

   a=3;
   b=4;
   sum=0;
   wa( a, b, sum );
   printf( "%d", sum);
   return 0;
}
```

wa 関数

```
void wa( int x, int y, int z ){
   z = x + y;
}
```

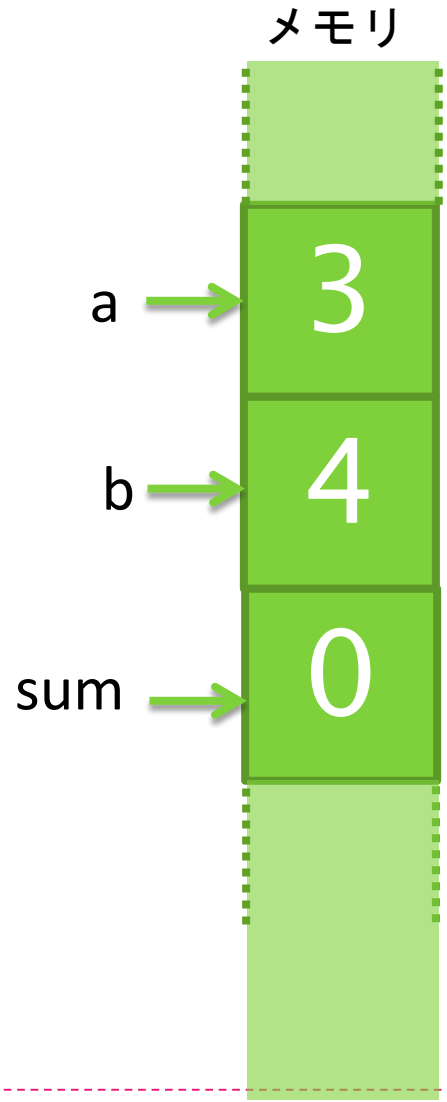
sum の値は？

0のまま！

メモリ上での関数の引数の挙動

main 関数

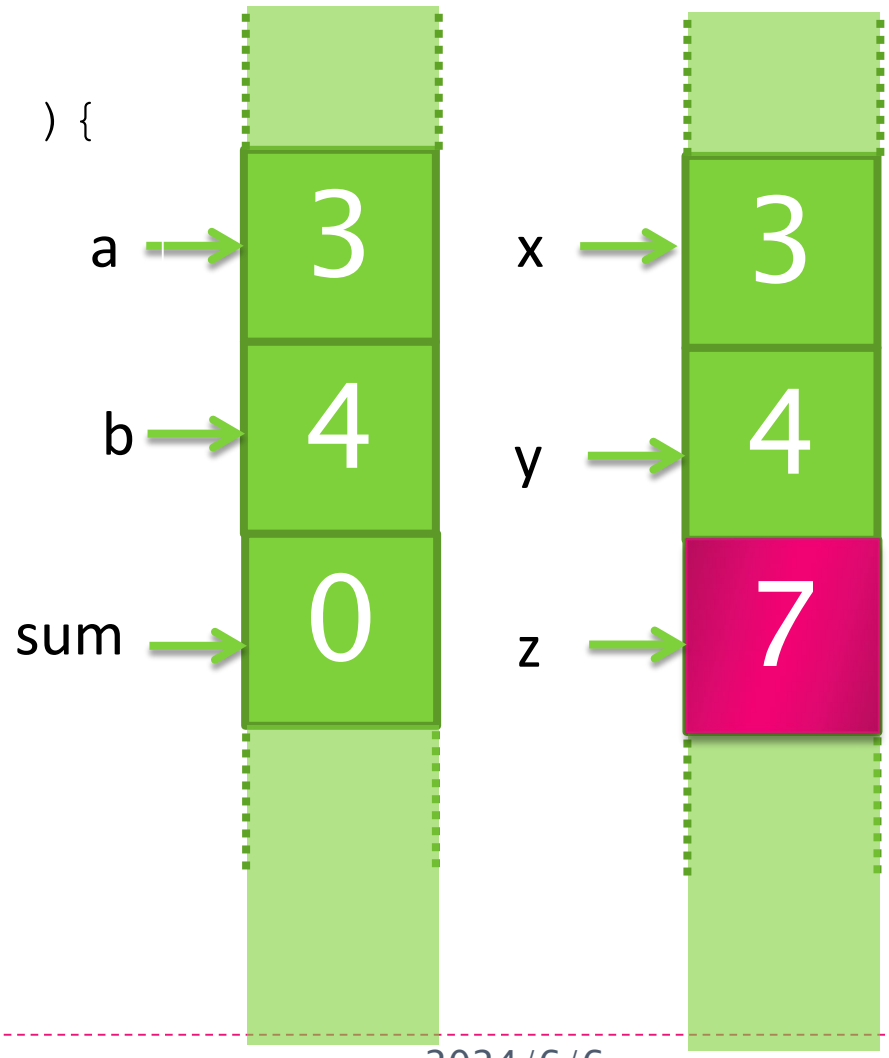
```
int main (void){

  int a, b, sum;

  a=3;
  b=4;
  sum=0;
  wa( a, b, sum );
  printf( "%d", sum);
  return 0;
}
```



メモリ上での関数の引数の挙動

wa 関数

```
void wa( int x, int y, int z ){
   z = x + y;
}
```

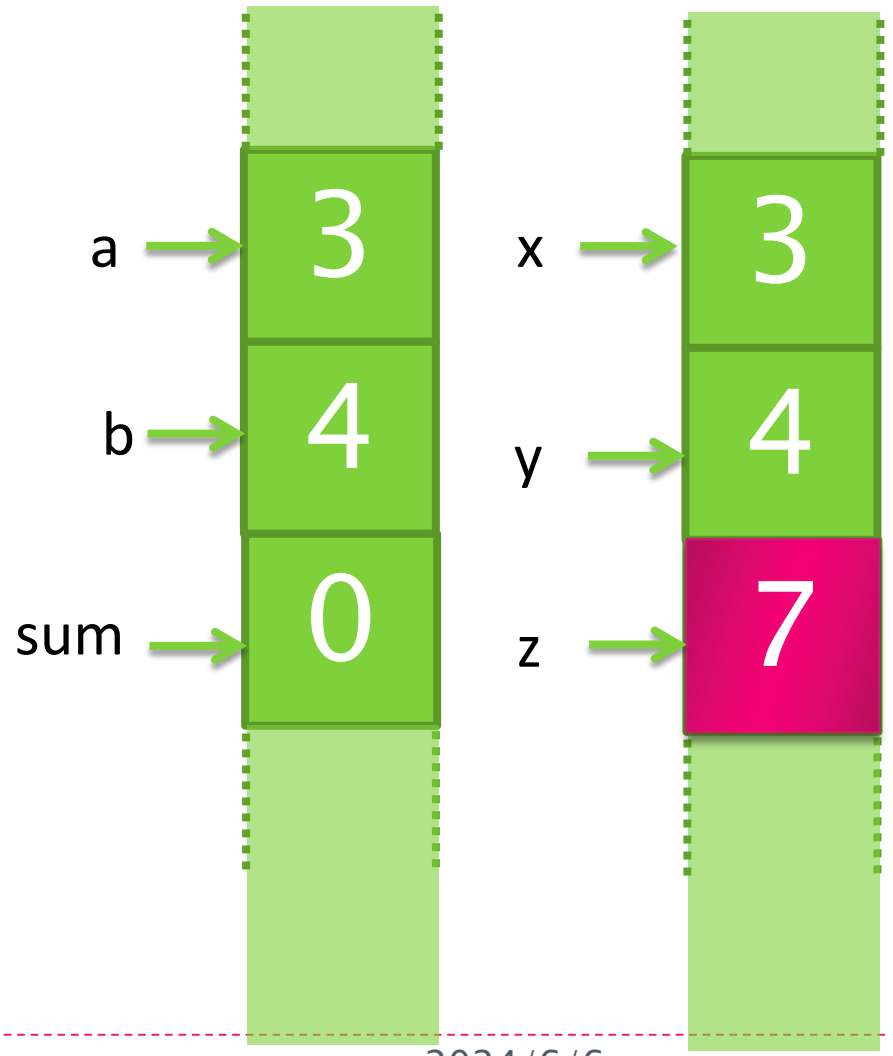


メモリ上での関数の引数の挙動

main 関数

```
int main (void){

  int a, b, sum;

  a=3;
  b=4;
  sum=0;
  wa( a, b, sum );
  printf( "%d", sum);
  return 0;
}
```

0のまま！



ポインタの利用

main 関数

```
int main (void){

  int a, b, sum;

  a=3;
  b=4;
  sum=0;
  wa( a, b, &sum );
  printf( "%d", sum);
  return 0;
}
```

wa 関数

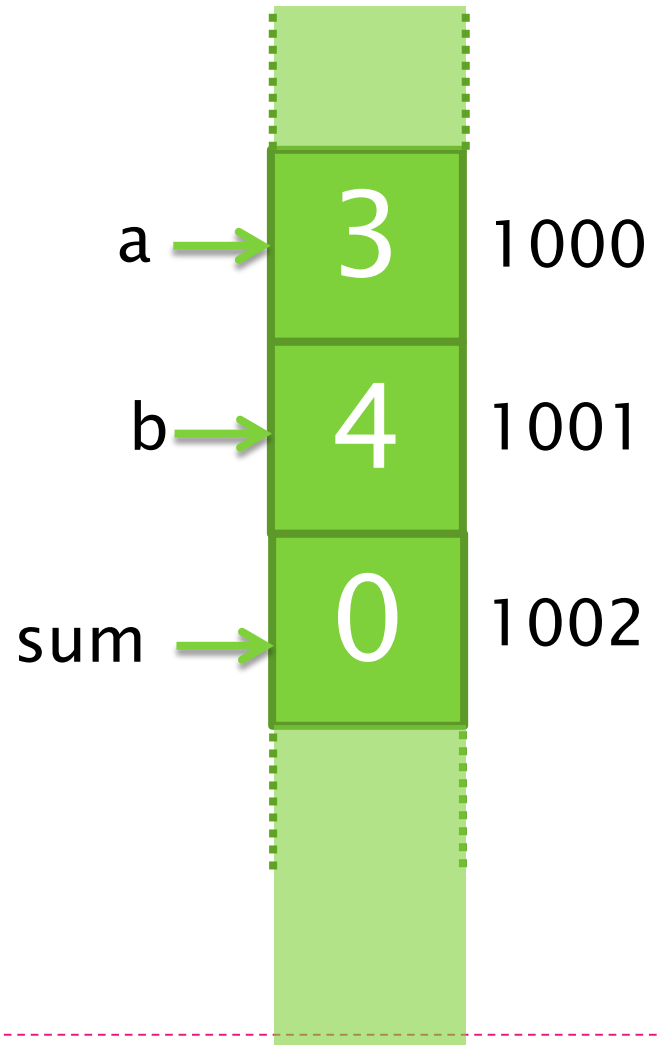
```
void wa( int x, int y, int* z ){
  *z = x + y;
}
```

7に変化

メモリ上の挙動

main 関数

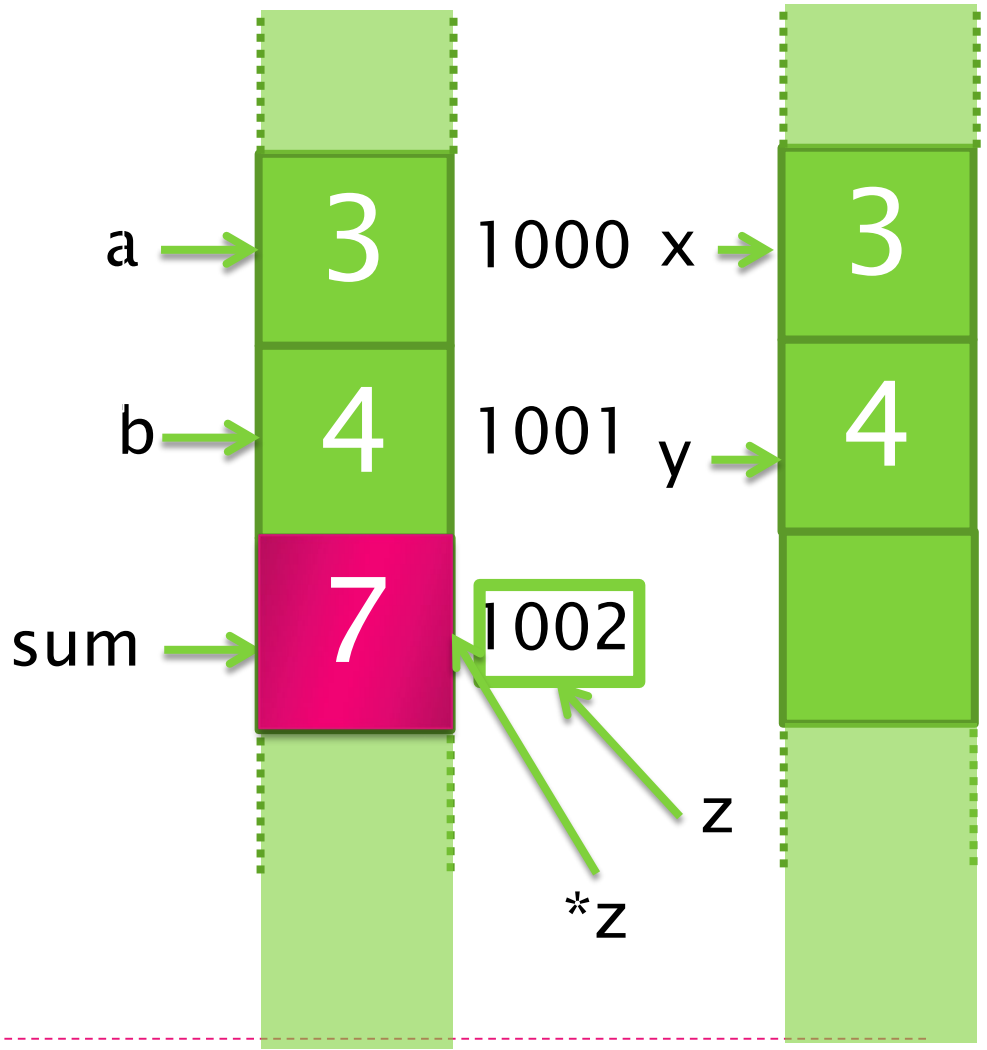
```
int main (void){

   int a, b, sum;

   a=3;
   b=4;
   sum=0;
   wa( a, b, &sum );
   printf( "%d", sum);
   return 0;
}
```



メモリ上での関数の引数の挙動

wa 関数

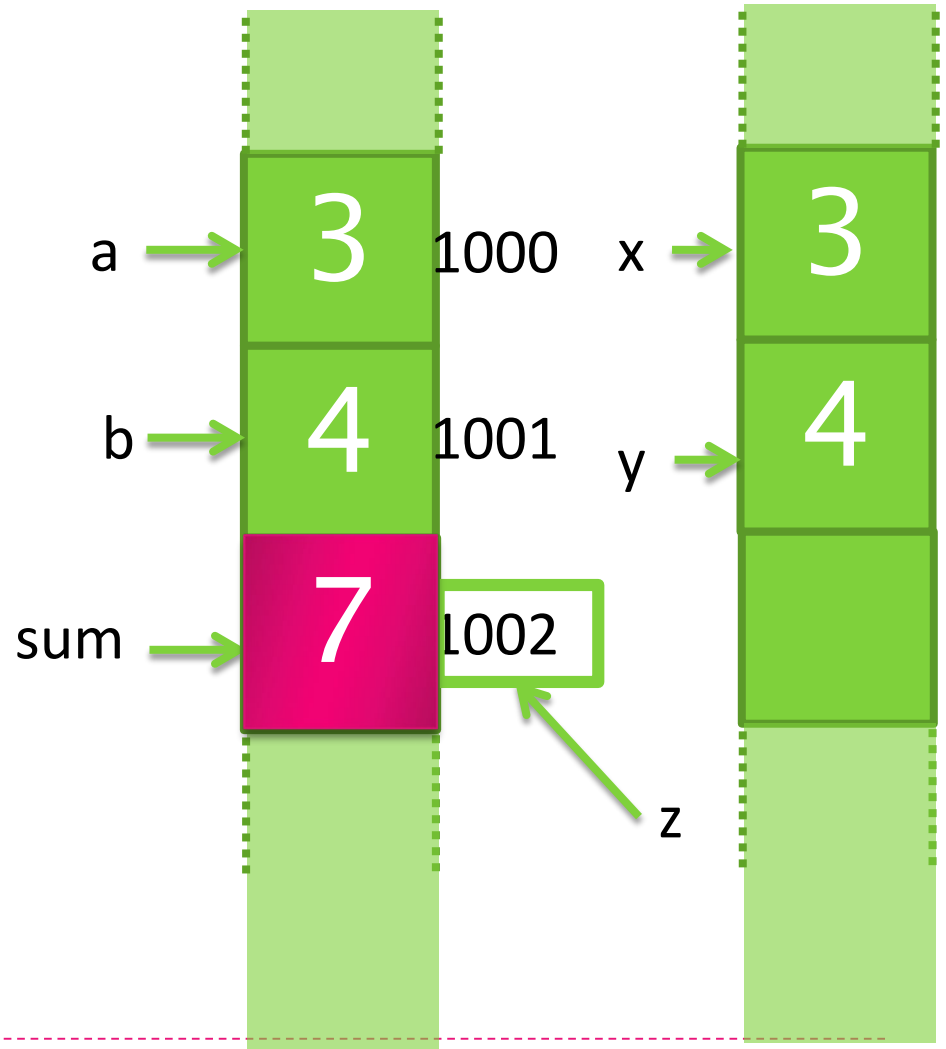
```
void wa( int x, int y,
int* z ){
   *z = x + y;
}
```



メモリ上での関数の引数の挙動

main 関数

```
int main (void){

   int a, b, sum;

   a=3;
   b=4;
   sum=0;
   wa( a, b, &sum );
   printf( "%d", sum);
   return 0;
}
```



演習課題（１）

1. ２つの double 型変数の四則演算を行う関数の定義を１つにまとめたファイル，プロトタイプ宣言をまとめたファイル，これらの関数を用いた main 関数があるファイルをそれぞれ作成し，分割コンパイルをして，プログラムを実行し，挙動を確認せよ．
   - 加算，減算，乗算，除算の4つの関数を作る
   - 関数のdouble 型仮引数を２つとし，戻り値を計算結果とする
2. キーボードから配列の要素数を入力し，malloc 関数でメモリを確保することで，ポインタを用いて int 型配列を実現せよ．また，配列に 0 から 99 までの乱数を代入し，これらの最大値，最小値，平均値，分散，中央値を求め，コマンドラインに出力せよ．

演習課題（２）

3. 授業支援システムにある kadai8-3.c は，関数 `replace` で２つの引数 `a,b` の値を入れ替えようとしている．しかし，このままでは値を入れ替えることは出来ない．プログラムの一部を修正して，値を入れ替え可能にせよ．また，レポートに値を入れ替えられるようになった理由を書け．
   - 修正したプログラムを提出すること
   - ファイルの名前を提出用に変更しておいて下さい

演習課題（３）

4. 授業支援システムにある kadai8-4.cというコードを実行すると，実行結果は以下の通りとなった.

```
pt_c: 0x7ffe3dce020b, 55
pt_i: 0x7ffe3dce020c, 1234
pt_c2: 0x7ffe3dce020c, -46
```

コードを見ると，`pt_c2` は `pt_i` を代入したものである．しかし，実行結果から，アドレスの値は同じになったものの，数値が変わって出力された．これはどうしてか．レポート内で理由を述べよ.

提出に関して(1/3)

- 提出するもの
  - ソースファイル(.c ファイル)（課題1, 2, 3 のみ）
    - ファイル名はkadai8-学籍番号-課題番号.c
      （課題1のmain 関数があるプログラムは
      kadai8-学籍番号-1-main.c，関数の定義はkadai8-学籍番号-1.c）
  - 課題1のヘッダファイル
    - ファイル名はkadai8-学籍番号-1.h
  - ソースファイル，ヘッダファイルにはコメントアウトで学籍番号と氏名を記入

提出に関して(2/3)

- 提出するもの（続き）
  - 実行結果の出力，課題 3, 4 の解答，講義へのコメント
    - TeX で作成した PDF ファイルで，レポート中に学籍番号，氏名を含む
    - ファイル名は report8–学籍番号.pdf
  - 全提出ファイルを 1 つのフォルダに格納し，そのフォルダを zip 圧縮したファイルを提出せよ.
    - フォルダ名：report8–学籍番号
    - zip 圧縮したファイル名: report8–学籍番号.zip

提出に関して(3/3)

- 提出期限
  - 6 月 12 日（水）　23：59
- 提出方法
  - Moodle から提出
- 注意点
  - ファイル名の命名規則が間違っているものは減点する
  - Repl.it の C モードで動作しないものは採点しない (0 点)