

2012年度 計算機システム演習  
第2回  
2012.04.20

白幡 晃一

# はじめに (再掲)

---

- ▶ 講義・演習のページ
  - ▶ 松岡研のHPからのリンク
    - ▶ <http://matsu-www.is.titech.ac.jp/lecture/lecture-wiki/> > 計算機システム (2012年度)
  - ▶ 内容: 授業のスライド、連絡事項等
  - ▶ 講義: 月曜3,4限, 演習: 金曜7,8限
  
- ▶ 演習内容 (講義 [前半]+ 計算機室で演習 [後半])
  - ▶ C言語プログラミングの基礎
  - ▶ MIPSシミュレータを用いたMIPSアセンブリプログラミング
  - ▶ MIPSシミュレータの作成
  - ▶ おまけ: PCの組み立て実習
  - ▶ 演習室:
    - ▶ 西7号館 3F演習室
  
- ▶ 質問等は...
  - ▶ 講義・演習の授業後
  - ▶ 松岡研究室: **西7号館 102号室**まで
  - ▶ メール
    - ▶ 白幡 晃一  
[koichi-s@matsulab.is.titech.ac.jp](mailto:koichi-s@matsulab.is.titech.ac.jp)

# メーリングリスト

---

- ▶ [compsys2012@matsulab.is.titech.ac.jp](mailto:compsys2012@matsulab.is.titech.ac.jp)
  - ▶ 事務連絡(休講情報, PC組立て演習, etc)
- ▶ 登録希望者は以下の内容で送信
  - ▶ 4/20(金)まで

To: 白幡 晃一 koichi-s@matsulab.is.titech.ac.jp

Subject: 【計算機システム】ML登録

----- (以下は本文に記述)

名前:

学籍番号:

登録メールアドレス:

※ ML削除⇒ 【計算機システム】 ML削除

# 課題の補足

---

## ▶ 課題1-1

▶ rand() の最大値は RAND\_MAX (stdlib.hで定義)

▶ `double x = rand() / RAND_MAX` → x の値は 0

▶ 整数同士の除算では1以下の結果は 0 と扱われるため

▶ 浮動小数点にキャストする必要あり

## ▶ ランダム関数の出力内容を実行毎に変える方法

▶ `srand((unsigned int)time(NULL))` (#include <time.h> が必要)

□ `time(NULL)` で現在時刻を取得し、`srand` で乱数系列のseedを与える

□ `unsigned int` 型 : 符号なしの整数値

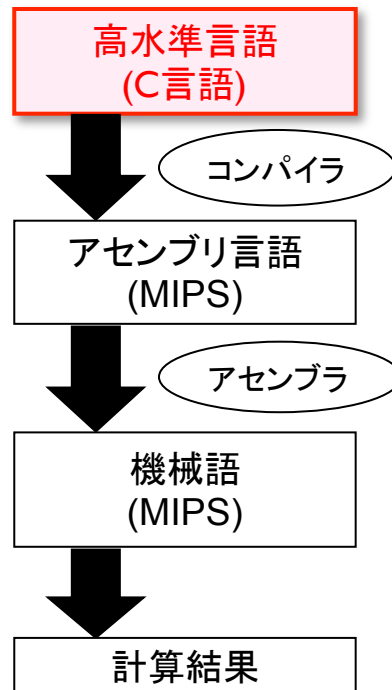
---

▶

# 課題の補足: emacs の使い方

- ▶ ショートカットの補足 → C: control、M: alt





## 今日の内容

構造体、連結リスト、TSUBAME

# 復習：ポインター

---

	d	&d	*d
int d (intを保持)	dの値 (つまりintの値)	dのアドレス	
int *d (intの変数のアドレスを保持)	dの値 (dが指す変数のアドレス)	dのアドレス	dが指す変数の値

※ \*：アドレスの参照先の値を出力



# (id, height)のペアをバブルソートするプログラム

sample19.c

```
#include <stdio.h>

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

void sort(int id[], int data[], int n){
    int k = n - 1;
    while ( k >= 1){
        int i;
        for (i = 1; i <= k; i++){
            if (data[i-1] > data[i]) {
                swap(&id[i-1],&id[i]);
                swap(&data[i-1],&data[i]);
            }
        }
        k -= 1;
    }
}
```

```
int main(){
    int i;
    int len = 5;
    int id[] = {100,101,102,103,104};
    int height[] = {184,164,175,171,179};
    sort(id, height, len);
    for (i = 0; i < len; i++){
        printf("%d:%d(id=%d)\n",i, height[i], id[i]);
    }
    return 0;
}
```

- ▶ 問題点
  - ▶ idとheightの関連性がわかりづらい
  - ▶ 例えばweightを追加したいときswapも追加
- ▶ javaならStudentの配列を作成
  - ▶ クラス名 : Student
  - ▶ フィールド : int id, int height



# 構造体

- ▶ 関連する個々のデータ (メンバ) をまとめて1つのデータとして扱える
  - ▶ Javaにおけるフィールドだけのクラス

- ▶ 定義方法

```
struct data {  
    char *name;  
    int val;  
};
```

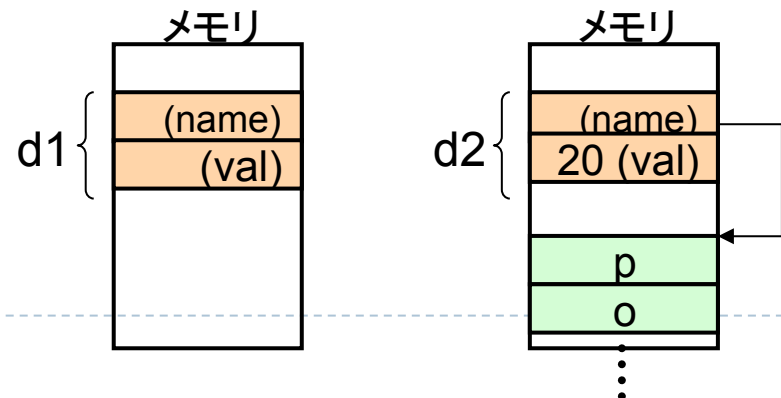
注意

- ▶ 関数の外に定義
  - ▶ スコープ: 全ての関数で使用できる
- ▶ 関数内に定義
  - ▶ スコープ: その関数のみで使用

- ▶ 宣言&使用方法

- ▶ 各メンバにはドット演算子(.)を用いてアクセス

```
struct data d1;  
struct data d2 = { "poteto", 20 };  
d1.name = "tomato";  
d1.val = 10;  
printf("%s %d\n",  
        d1.name, d1.val);
```



# サンプル：構造体の使用方法

sample18.c

```
#include <stdio.h>
struct student {
    char *name;
    int height;
    double weight;
};
int main(){
    struct student sdt1 = {"ichiro", 180, 68.5};
    struct student sdt2;
    sdt2.name = "hanako";
    sdt2.height = 170;
    sdt2.weight = 60.2;

    printf("std1=%s, %d, %f\n", sdt1.name, sdt1.height, sdt1.weight);
    printf("std2=%s, %d, %f\n", sdt2.name, sdt2.height, sdt2.weight);
}
```

std1=ichiro, 180, 68.500000  
std2=hanako, 170, 60.200000

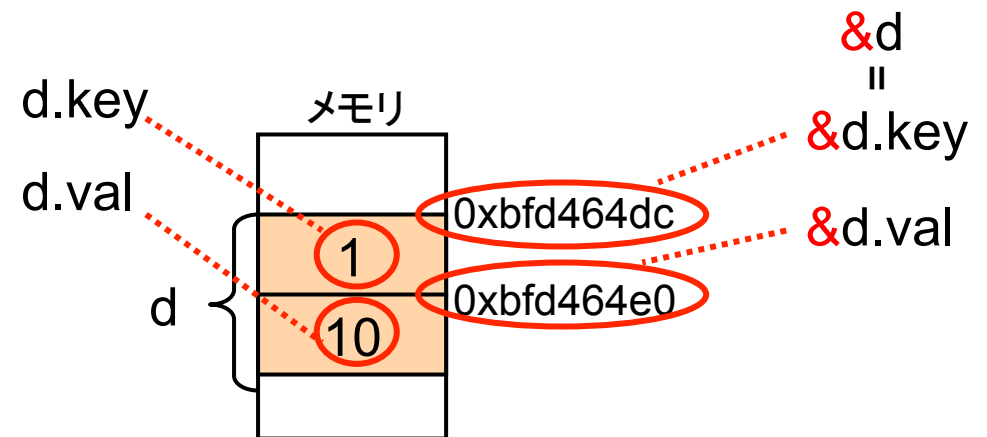
# 構造体のメモリ配置

- ▶ 各メンバは宣言された順にメモリ上に配置
  - ▶ 各々アドレスを持つ

```
struct student {  
    int key;  
    int val;  
};  
int main() {  
    struct data d = { 1, 10 };  
    printf("&d.key=%p\n", &d.key);  
    printf("&d  =%p\n", &d);  
    printf("&d.val=%p\n", &d.val);  
    return 0;  
}
```

Output

```
&d.key=0xbf464dc  
&d  =0xbf464dc  
&d.val=0xbf464e0
```



注) &d.key = &(d.key)

結合度: . > &

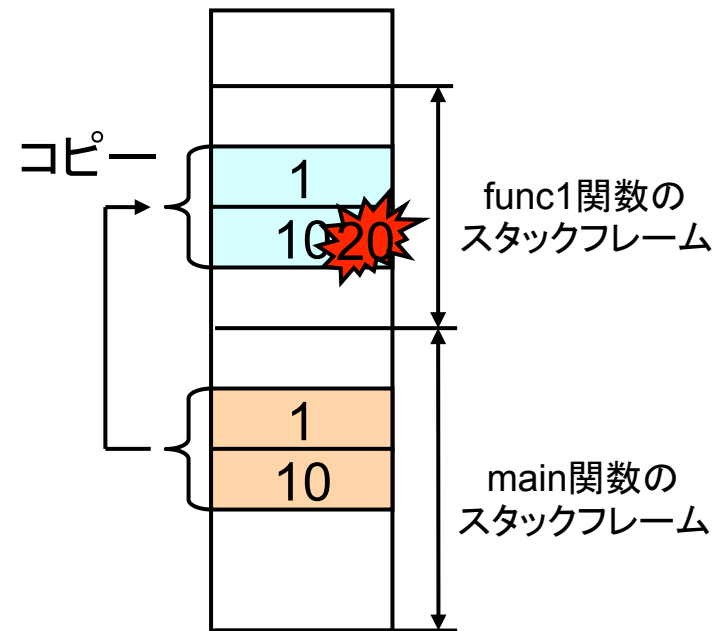
# 構造体と関数

## ▶ 構造体は値渡し

- ▶ 関数内で値を変更しても、呼び出し元は変更されない

sample21.c

```
#include <stdio.h>
struct data {
    int key;
    int val;
};
int func1(struct data d) {
    d.val = 20;
}
int main() {
    struct data d = {1, 10};
    printf("d.val = %d\n", d.val);
    func1(d);
    printf("d.val = %d\n", d.val);
    return 0;
}
```



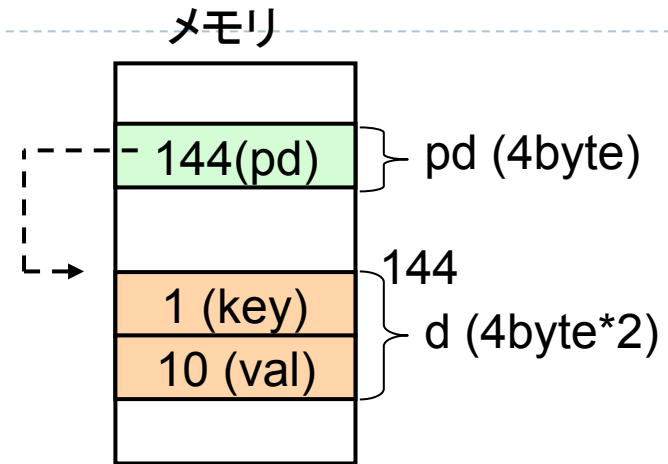
出力

```
d.val = 10
d.val = 10
```

# 構造体とポインタ

- ▶ ポインタ変数の宣言、初期化

```
struct data d = { 1, 10 };  
struct data *dp;  
dp = &d;
```



- ▶ ポインタを介したメンバへのアクセス

- ▶ アロー演算子(→)を使用

- ▶ dp->val と (\*dp).val は同じ (記述の簡略化のため)

```
printf("( *dp ).val = %d\n", (*dp).val); //10  
printf("dp->val = %d\n", dp->val); //10  
dp->val = 100;  
printf("dp->val = %d\n", dp->val); //100  
printf("d.val = %d\n", d.val); //100
```

← 結合度: . > \*

# サンプル：構造体とポインタ

sample22.c

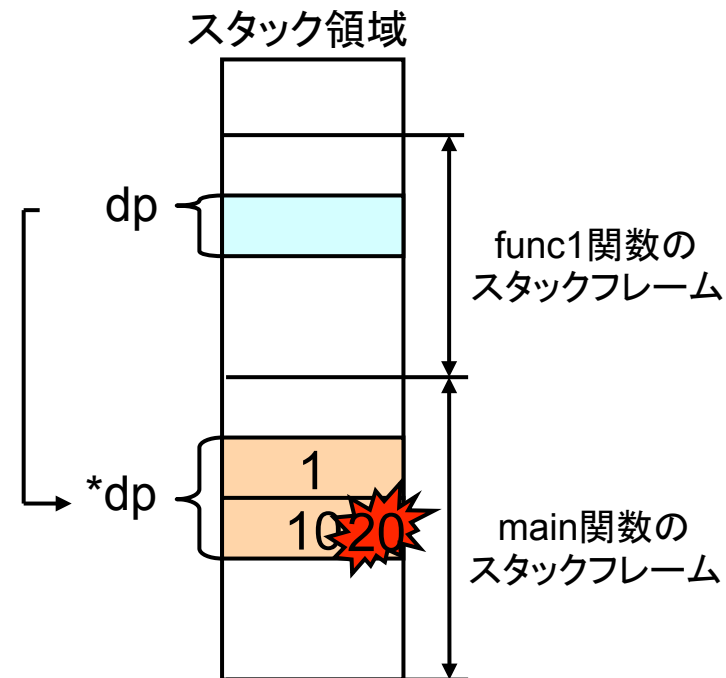
```
#include <stdio.h>

struct data {
    int key;
    int val;
};

int func1(struct data *d) {
    d->val = 20; // (*d).val=20
}

int main() {
    struct data d = {1, 10};
    printf("d.val = %d\n", d.val);
    func1(&d);
    printf("d.val = %d\n", d.val);
    return 0;
}
```

- ▶ ポインタを用いて、呼び出しもとの値を変更



```
d.val = 10
d.val = 20
```

# (id, height)のペアをバブルソートするプログラム2

sample20.c

```
#include <stdio.h>
struct data {
    int id;
    int height;
};
void swap(struct data *dx, struct data *dy)
{
    struct data d = *dx;
    *dx = *dy;
    *dy = d;
}
void sort(struct data d[], int n)
{
    int k = n - 1;
    while ( k >= 1){
        int i;
        for (i = 1; i <= k; i++){
            if (d[i-1].height > d[i].height) {
                swap(&d[i-1],&d[i]);
            }
        }
        k -= 1;
    }
}
```

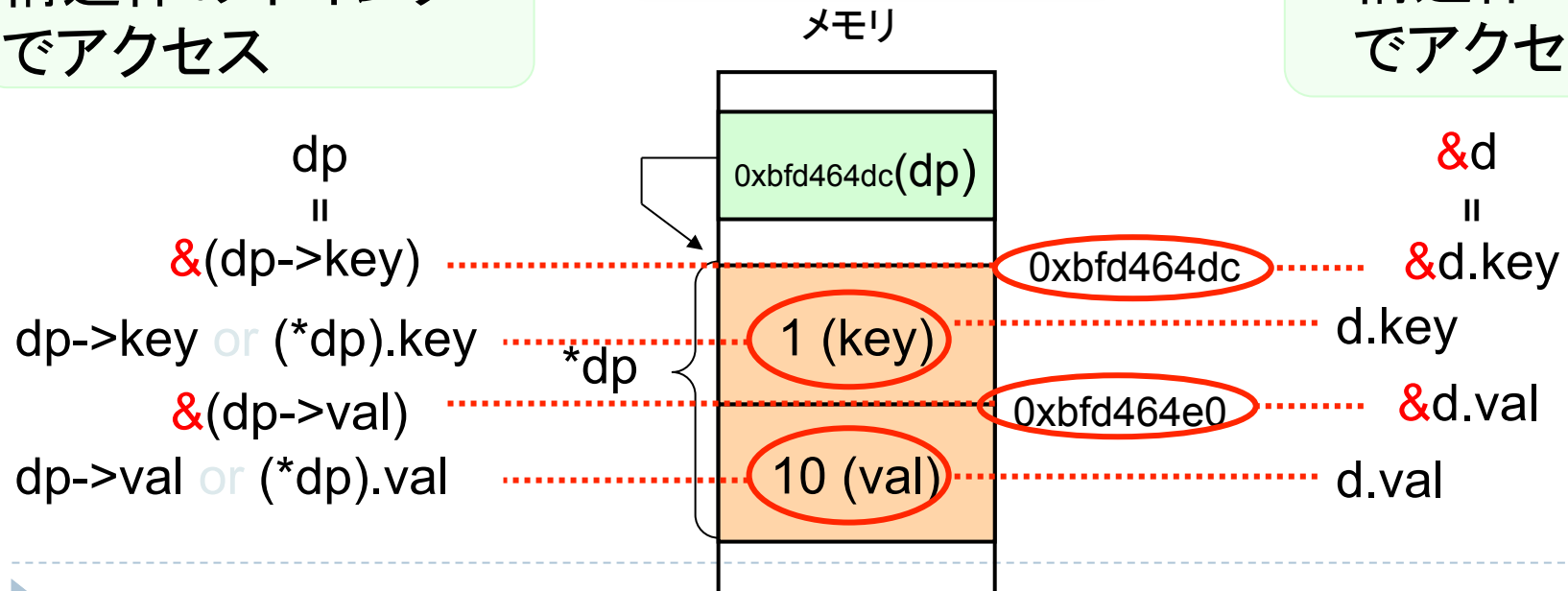
```
int main()
{
    int i;
    int len = 5;
    struct data d[] = {
        {100, 184},
        {101, 164},
        {102, 175},
        {103, 171},
        {104, 179}
    };
    sort(d, len);
    for (i = 0; i < len; i++)
        printf("%d:%d(id=%d)\n",i, d[i].height, d[i].id);
    return 0;
}
```

# 構造体とポインタ (まとめ)

```
struct data {  
    int key;  
    int val;  
};  
  
struct data d = {1, 10};  
struct data *dp;  
dp = &d;
```

構造体のポインタ  
でアクセス

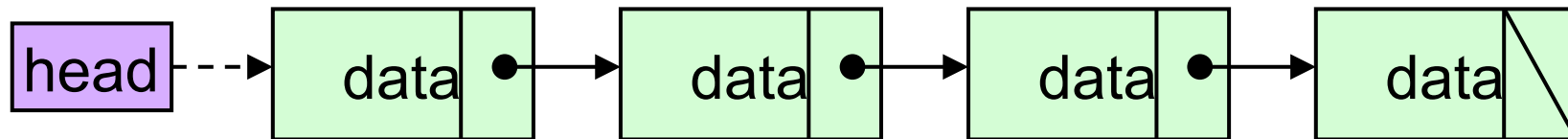
構造体  
でアクセス





# 構造体の活用： 連結リスト

---



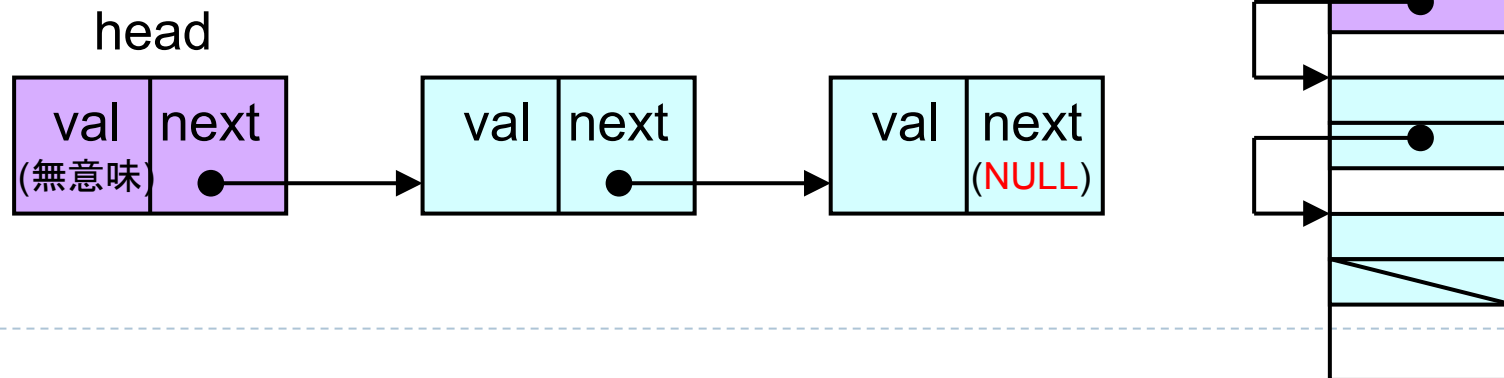
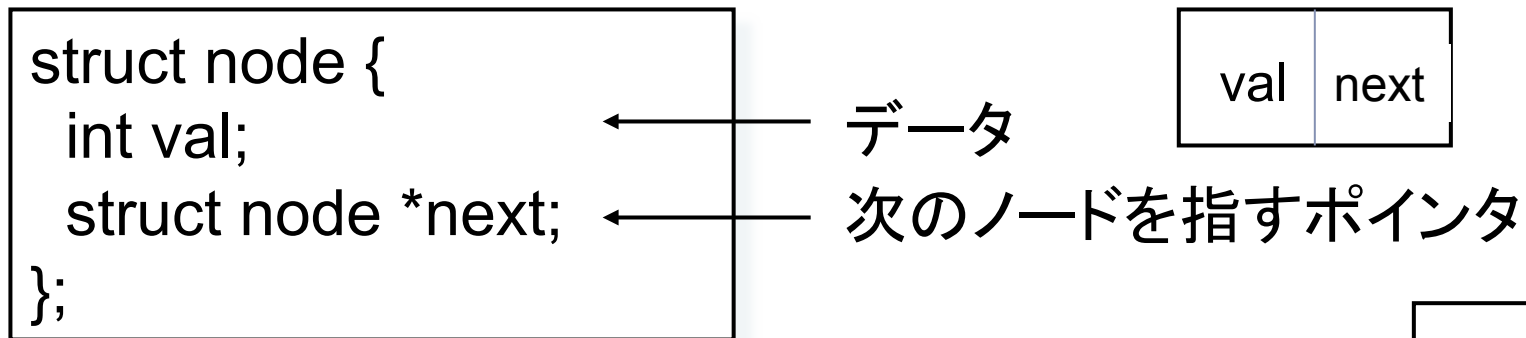
- ▶ 構造体の使用例
  - ▶ 連結リスト、ツリー構造、キュー、スタック
- ▶ データをリンクで数珠繋ぎにしたリスト
  - ▶ 連結リストを構成する各要素をノードと言う



# ノードのデータ構造

## ▶ 構造体で定義

- ▶ headは先頭ノードを指す特別なnode構造体型データ



# 連結リストの基本的な関数

---

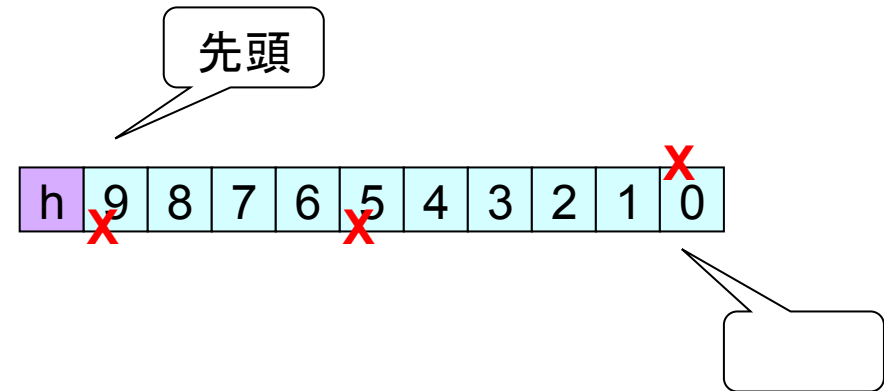
- ▶ `int add (struct node *head, int val);`
  - ▶ head が指すリストの先頭に val を追加
  - ▶ 追加に成功したら0を、失敗したら-1を返す
- ▶ `int deleteFirst (struct node *head);`
  - ▶ head が指すリストから先頭要素 (headの次のノード)を削除
  - ▶ リストが空なら-1を、空でなければ先頭要素の値を返す
- ▶ `int delete (struct node *head, int val);`
  - ▶ head が指すリストから要素 val を削除
  - ▶ val がリスト中にあれば val を、無ければ-1を返す
- ▶ `void display (struct node *head);`
  - ▶ head が指すリスト中の要素を全て表示



# 使い方

sample23.c

```
int main() {
    struct node head = {-1, NULL};
    int nums[] = {9,8,7,6,5,4,3,2,1,0};
    int i;
    for (i = 0; i < 10; i++){
        int res = add(&head, nums[i]);
        if (res != 0) return 1;
    }
    deleteFirst(&head); // 9が削除
    delete(&head, 9); // -1を返す
    delete(&head, 5);
    delete(&head, 0);
    display(&head);
    return 0;
}
```



出力

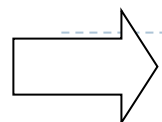
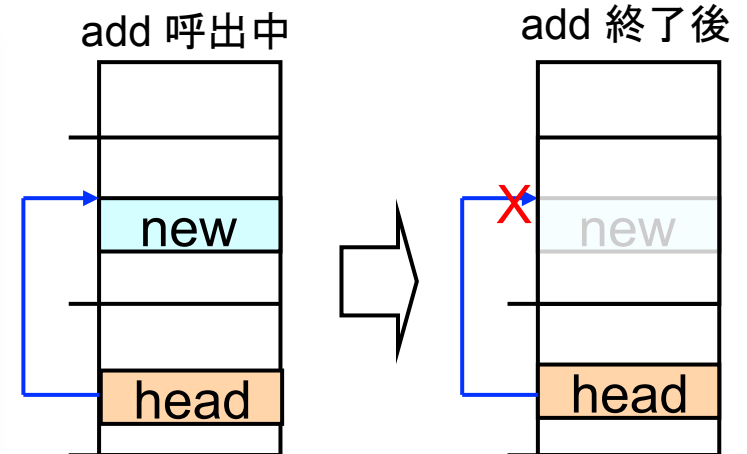
```
1: 1
2: 2
3: 3
4: 4
5: 6
6: 7
7: 8
```

# add関数 (悪い例)

- ▶ ローカル変数 (struct node new) では使えない
  - ▶ newはスタック領域上のadd関数のスタックフレーム内に作られる
    - ▶ add終了時にはスタックフレームごと削除される

```
int add(struct node *head, int val) {  
    struct node new;  
  
    new.val = val;  
    new.next = head->next;  
    head->next = new;  
    return 0;  
}
```

ダメ



確保したメモリ領域がadd関数後も削除されないようにする必要がある

# malloc, free 関数

---

- ▶ `void *malloc(size_t size)`
  - ▶ ヘッダファイル `stdlib.h` (`#include <stdlib.h>`)
  - ▶ `size` **バイト**分のメモリ領域を確保し、確保した領域の先頭のアドレスを返す
    - ▶ **ヒープ領域**上に確保する⇒プログラム実行中消えることはない
    - ▶ `size_t`型は多くの処理系で「`unsigned long (int)`」型
  - ▶ ポインタさえ取得できれば全ての関数からアクセス可能
  - ▶ `void`ポインタはどのType型にも代入可能
    - ▶ 一般に代入される型にキャストする
  - ▶ `java`の`new Object()`みたいなもの
- ▶ `void free(void *ptr)`
  - ▶ ヘッダファイル `stdlib.h`
  - ▶ `ptr`が指すヒープ領域上のメモリ領域を解放

⇒ ヒープ領域上のデータは明示的に削除しないと、プログラム終了時までメモリを消費し続ける。  
(開放し忘れを「メモリリーク」という)

  - ▶ `java`ではGC(ガベージコレクター)が使用されなくなったメモリ領域を解放している。



# 型のサイズ

---

## ▶ sizeof演算子を用いて取得

▶ `size_t size = sizeof(Type)`

▶ Type型のサイズをバイト単位で返す

▶ size\_t型は多くの処理系で「unsigned long (int)」型

```
struct data {
    int key;
    int val;
};
int main() {
    size_t size = sizeof(struct data);
    printf("%d\n", size);
    printf("%lu\n", (unsigned long)size);
    return 0;
} // ともに8を出力
```

小さいサイズであれば  
int型として表示もできる

正しい(安全)書き方

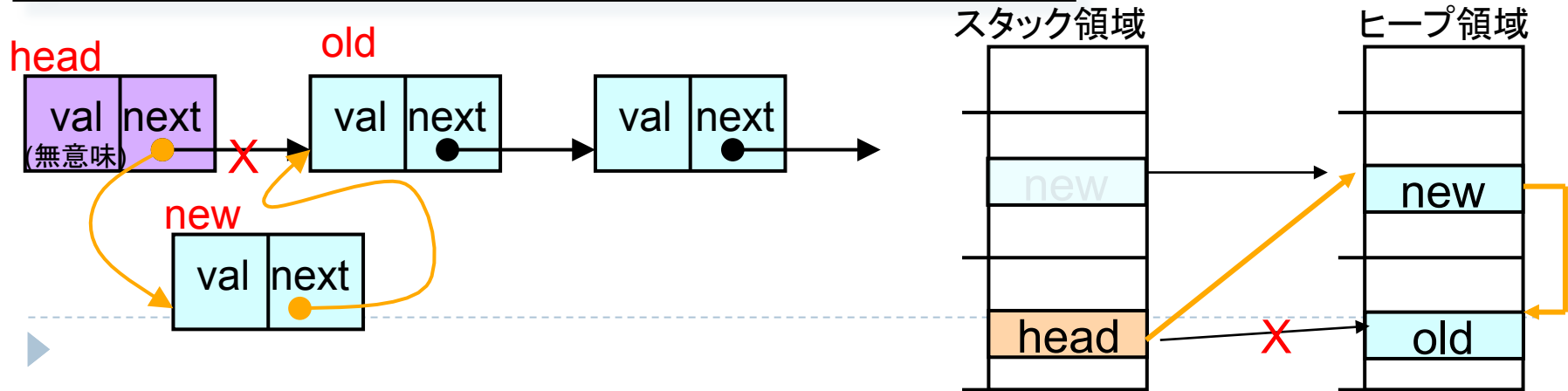
# add 関数

sample23.c

```
int add(struct node *head, int val) {  
    struct node *new;  
    new = (struct node *)malloc(sizeof(struct node));  
    if (new == NULL) return -1;  
  
    new->val = val;  
    // head->nextがNULLかチェック  
    new->next = head->next;  
    head->next = new;  
    return 0;  
}
```

メモリ確保

確保されているかしっかりチェックすること

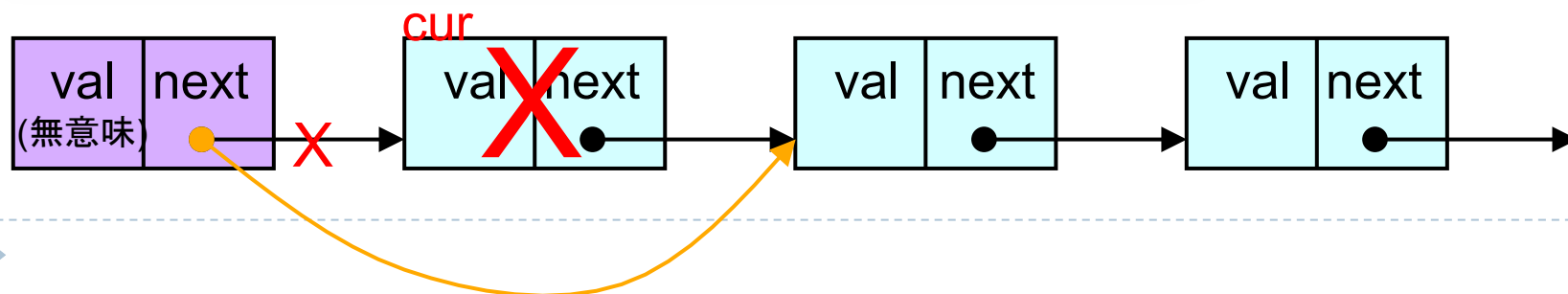




# deleteFirst 関数

sample23.c

```
int deleteFirst(struct node *head) {  
    struct node *cur;  
    int result = -1;  
    if (head->next != NULL) {  
        cur = head->next;  
        head->next = cur->next;  
        result = cur->val;  
        free(cur);  
    }  
    return result;  
}
```



# delete 関数

---

sample23.c

```
int delete(struct node *head, int val) {
    struct node *cur, *prev;
    int result = -1;
    // head->next == NULL check
    for (cur = head->next, prev = head; cur != NULL;
         cur = cur->next, prev = prev->next) {
        if (cur->val == val) {
            prev->next = cur->next;
            result = cur->val;
            free(cur);
            break;
        }
    }
    return result;
}
```

# typedef

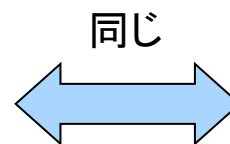
---

- ▶ 既存の型に対して同義語を与える宣言

```
typedef <既存型> <新規型>;
```

- ▶ 例) `typedef int NUMBER;` `typedef unsigned long size_t`
- ▶ メリット1: 読みやすさ・書きやすさ向上
  - ▶ 毎回 `struct data` と書く必要がない

```
struct _data {  
    int key;  
    int val;  
};  
typedef struct _data data;
```



```
typedef struct {  
    int key;  
    int val;  
} data;
```

- ▶ メリット2: コードに影響を与えず、既存型を置き換えられる
    - ▶ `typedef int NUMBER;` ⇒ `typedef double NUMBER;`
- 



# typedef (例)

sample31.c

```
#include <stdio.h>
struct _map1 {
    int key;
    int value;
};
typedef struct _map1 map1;

typedef struct {
    int key;
    int id;
    int value;
} map2;

int main() {
    map1 m1 = {1, 10};
    map2 m2 = {2, 2, 20};
    printf("m1={%d, %d}\n", m1.key, m1.value);
    printf("m2={%d, %d, %d}\n", m2.key, m2.id, m2.value);
}
```

```
m1={1, 10}
m2={2, 2, 20}
```

# #define

---

sample32.c

```
#include <stdio.h>
#define LENGTH 5

int main()
{
    int i;
    int nums[LENGTH] = {0,1,2,3,4};

    for (i=0; i < LENGTH; i++){
        printf("%d\n",nums[i]);
    }
    return 0;
}
```

- ▶ コンパイル前に指定した値に置換する

```
#define <記号定数名> <値>
```

- ▶ #define NUMBER 5
- ▶ 利点
  - ▶ 値の管理を1箇所に集約
  - ▶ プログラム中の数字(マジックナンバー)に名前を与えることで可読性向上



# TSUBAME2.0上でのジョブ実行

---

## ▶ 目標

- ▶ プログラムをTSUBAME上で動かしてみる

## ▶ 概要

- ▶ step 1: sshでTSUBAMEにログイン
- ▶ step 2: scpでTSUBAMEにファイル転送
- ▶ step 3: TSUBAMEに実行したいjobをサブミット
- ▶ step 4: 実行結果の確認

## ▶ HP

- ▶ <http://tsubame.gsic.titech.ac.jp/login>



## step 1: sshでTSUBAMEにログイン

---

- ▶ SSH (Secure Shell)とは
  - ▶ 遠隔地 (リモート)のコンピュータにネットワークを介してセキュアにログイン・通信するためのプロトコル
  - ▶ sshではネットワーク上の通信を暗号化
  
- ▶ 似たようなプロトコル
  - ▶ telnet, rsh, rlogin, etc
  - ▶ いずれもパスワードが平文でネットワーク上を流れるため安全じゃない



# step 1: sshでTSUBAMEにログイン

- ▶ ターミナルを起動 → ssh コマンド

```
> ssh <教育用PCのアカウント>@login-t2.g.gsic.titech.ac.jp  
passwd: XXXXXXXX (パスワードを打っても何も表示されない)
```

-----  
please refer the following page.

<http://tsubame.gsic.titech.ac.jp/ja/node/274>

```
-----  
<教育用PCのアカウント>@t2a006167:~> |
```

Cygwin (terminal)を起動してコマンドを実行

- ▶ TSUBAMEではログイン用のサーバー
  - ▶ login-t2.g.gsic.titech.ac.jp
- ▶ ログイン後、自動的にインタラクティブマシンへ転送



## step 2: scpでファイル転送

---

### ▶ scpとは

- ▶ リモートのマシンに対してsshを用いてファイルを転送するプログラム
- ▶ イメージとしてはcp + ssh
- ▶ scp 転送したいファイル
  - ▶ [user-ID@login-t2.g.gsic.titech.ac.jp](#):<転送先のパス>

TSUBAME側

```
>pwd  
/home/user6/satou-k-ab (人によって違うので注意)
```

Terminal をもう1つ起動してscpで転送

クライアント側

```
> cd Kadai/ex02/  
> scp ex02.c user-ID@login-t2.g.gsic.titech.ac.jp:/home/  
user6/satou-k-ab
```



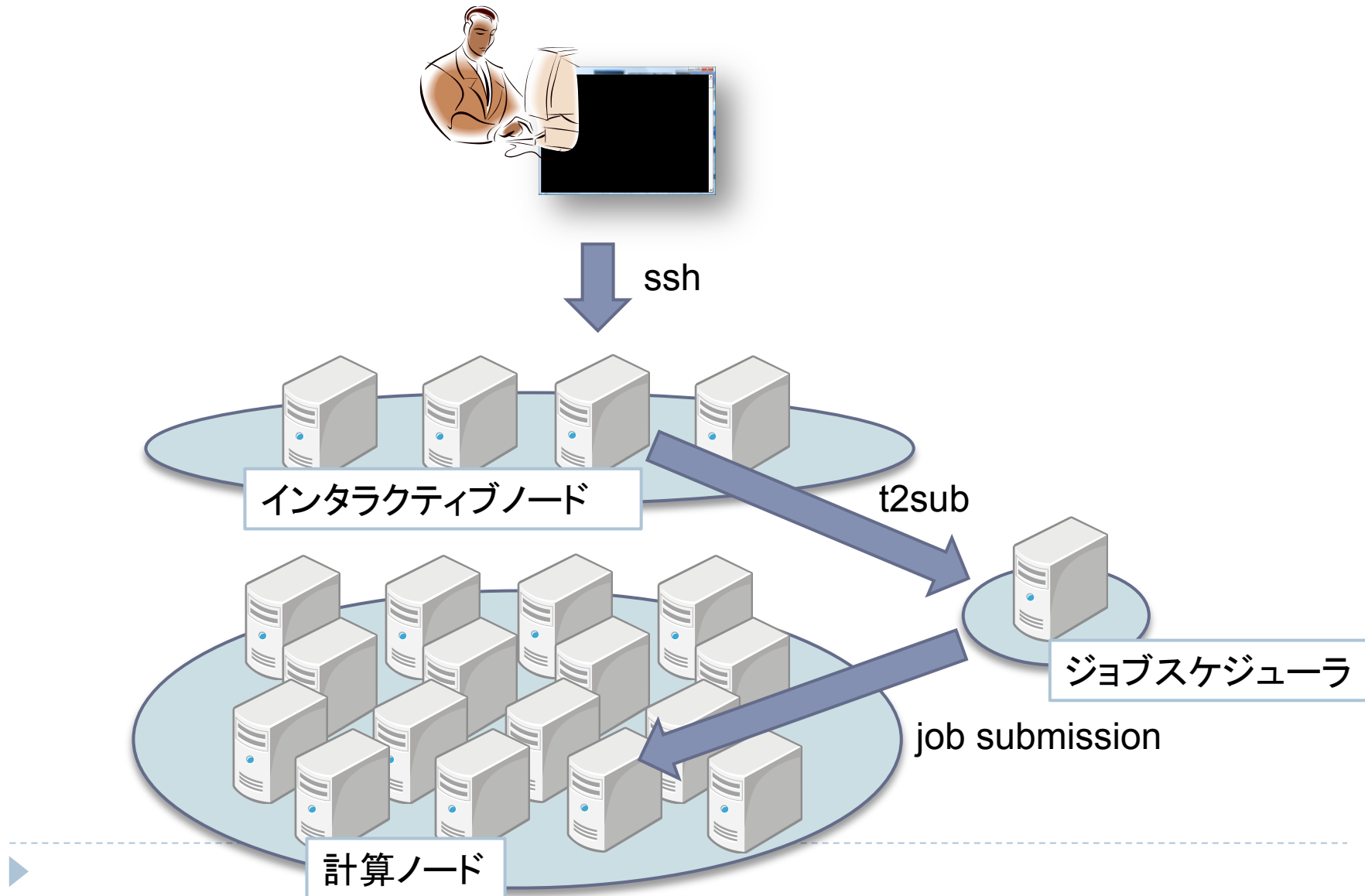
## step 3: TSUBAMEへjobのサブミット(1/2)

---

- ▶ バッチ処理システムを利用
  - ▶ インタラクティブに個々のユーザが利用すると・・・？
    - ▶ → リソースの競合が起こる
  - ▶ バッチキューシステムを利用することで、システムがジョブをスケジューリング
    - ▶ → TSUBAME全体として効率よく資源を利用
- ▶ バッチキューシステム
  - ▶ t2sub (TSUBAMEで利用)
  - ▶ torque
  - ▶ qsub



# step 3: TSUBAMEへjobのサブミット(2/2)



# step 3: TSUBAMEへjobのサブミット

---

## 1. jobの作成

- ▶ シェルスクリプトで実行したいjobを記述

```
> emacs myjob.sh
```

```
#!/bin/sh  
cd /home/user6/satou-k-ab //実行したいプログラムがあるディレクトリ  
./ex02 //timeプログラムの実行時間を計る。詳しくは”man time”
```

## 2. 実行権を与える

```
> chmod 700 myjob.sh
```



## step 3: TSUBAMEへjobのサブミット

---

### 3. queueにjobをサブミット

```
> t2sub -N compsysJob -q S ./myjob.sh
Checking accounting informations...
Warning: Because no group id is specified, following limitation is applied.
available queue(s) : S,L128,L128F,L256,L512,S96
walltime limitation : 00:10:00
maximum available nodes : 2
Checking requested resources...
Submitting a job to PBS...
116136.t2zpbs01
```

### 4. 実行結果の確認

```
> ls
compsysJob.e116136 compsysJob.o116136
```

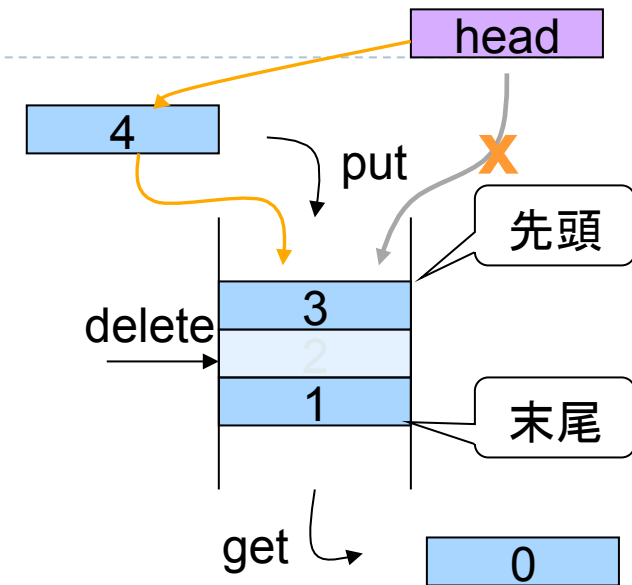
- ▶ compsysJob.eXXXXXXXX: エラー出力
- ▶ compsysJob.oXXXXXXXX: 標準出力 <= 実行結果の出力先



# 本日の課題

# 課題1

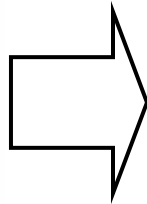
- ▶ 連結リストを参考にQueueを作れ
  - ▶ Queue: First In, First Outなリスト
  - ▶ 実装する関数
    - ▶ **int put (struct node \*head, int val)**
      - Queue(の先頭)にデータを追加(enqueue)
      - 戻り値: 成功=>0、失敗=>-1
    - ▶ **int get (struct node \*head)**
      - Queue(の末尾)からデータを取得&削除(dequeue)
      - 戻り値: データが存在する=>val、存在しない=>-1
    - ▶ **int delete (struct node \*head, int val)**
      - Queueから値がvalである要素を削除
      - 戻り値: データが存在する=>val、存在しない=>-1
    - ▶ **void display (struct node \*head)**
      - Queueの内容を“末尾”から順に表示
  - ▶ 動作を確認するためのmain関数も記述すること
    - ▶ put、getするデータの出力、displayによる出力を行い、実行結果もレポートにまとめること



# 課題1 : main関数例

---

```
struct node head = {-1, NULL};
int nums[] = {0,1,2,3,4,5,6,7,8,9};
int i, res;
for (i = 0; i < 10; i++) {
    printf("put %d\n", nums[i]);
    res = put(&head, nums[i]);
    if (res != 0) return 1;
}
display(&head);
for (i = 0; i < 3; i++) {
    res = get(&head);
    printf("get %d\n", res);
}
display(&head);
res = delete(&head, 7);
printf("delete %d\n", res);
display(&head);
```



```
put 0
put 1
put 2
put 3
put 4
put 5
put 6
put 7
put 8
put 9
queue: 0 1 2 3 4 5 6 7 8 9
get 0
get 1
get 2
queue: 3 4 5 6 7 8 9
delete 7
queue: 3 4 5 6 8 9
```



## 課題2: TSUBAME2.0上でのジョブ実行

---

- ▶ 行列積を行うプログラムを作成し、実行時間を比較せよ
  - ▶ グラフを作成比較せよ
    - ▶ Mac v.s. TSUBAME2.0 (1 node)
    - ▶ 縦軸: 実行時間、横軸: 問題サイズによる違い
- ▶  $A(10 \times 10) \times B(10 \times 10) = C(10 \times 10)$ の行列演算
  - ▶ `int matmul (int *a, int *b, int *c, int size)`
    - ▶ a, b: 入力行列
    - ▶ c: 解
    - ▶ **Size: 行列サイズ**
    - ▶ 戻り値: どうすればよいか?
- ▶ 実行方法

```
> ./matmul 10
```

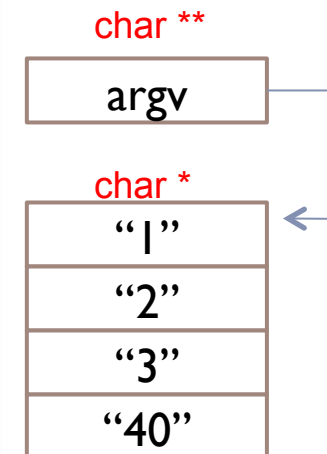
---

# ヒント: main関数の引数

- ▶ 文字列 (char \*) へのポインターは char \*\*

```
int main(int argc, char **argv) {
    int i;
    printf("argc=%d\n", argc); // 引数の数
    printf("bin=%s\n", argv[0]); // argv[0]は、実行ファイル名
    for (i = 1; i < argc; i++) {
        // stdlib.hに文字列=>int変換関数 atoi が定義
        printf("argv[%d]=%d\n", i, atoi(argv[i]));
    }
    return 0;
}
```

```
> ./a.exe 1 2 3 40
argc=5
bin=./a.exe
argv[1]=1
argv[2]=2
argv[3]=3
argv[4]=40
```



# ヒント： 行列積

---

- ▶ main 関数の引数
  - ▶ 文字列 (char \*) へのポインターは char \*\*

```
int main(int argc, char **argv) {  
    // 問題サイズ(行列のサイズ)を取得  
    // mallocを用いて行列の配列を格納する領域を確保  
    // 行列の初期化 (静的、rand など)  
    int r = matmul (a, b, c, size) // 入力行列 a, b 積 : c  
    return 0;  
}
```



## 課題提出(1/2)

---

- ▶ 〆切: **5/11(金) 23:59**
  - ▶ 遅れても受け付けます。(でも、減点あり)
- ▶ 提出物: 以下のファイルを**圧縮したもの**
  - ▶ ドキュメント (pdf,plain txt,word形式)
    - ▶ プログラムソースの簡単な説明、グラフ、工夫したところ
    - ▶ 感想、質問等
  - ▶ プログラムソース (課題1,2)
    - ▶ テスト用のmain関数も含む(コンパイルできて正しく実行できること)

# 課題提出(2/2)

## ▶ 提出方法: Webから提出

- ▶ パスワードは白幡まで
- ▶ アップローダーに問題がございましたら、至急白幡まで連絡を！


7/20(Fri)		第14回(予定)
7/23(Mon)	第13回(予定)	

### 課題関係 <sup>+</sup>

#### 課題関連 <sup>+</sup>

- 課題提出

#### 参考資料 <sup>+</sup>

- サンプル
  -  C言語

[\[Top\]](#)

## 計算機システム2012 (課題提出ページ)

### 演習課題

- 第1回 [\[提出\]](#) [\[提出状況\]](#) : C言語の基礎 1
  - 〆切: 2011/4/27 Fri 23:59
  - 提出物: 以下を圧縮して提出
    - ドキュメント (ソースコードについて、感想、質問等)
    - ソースコード