

ソフトウェア2 第2回 (2015/11/26)

鶴岡 慶雅

連絡用ページ

- URL

<http://www.logos.t.u-tokyo.ac.jp/~tsuruoka/lecture/software2/>

ユーザ名: ee2015

パスワード: soft2

- 資料

- 講義スライド

- サンプルプログラム

便利なコマンドなど

- シェル上で
 - TAB キーによるファイル名の補完
 - 上カーソルキーによるコマンドヒストリの利用
 - マニュアル閲覧
 - `include` が必要なヘッダーファイルもこれでわかる
 - 例 `man printf`
- シフトロック状態の解除
 - 常にシフトキーが押されているような状態になってしまったら
 - `[Fn] + [Shift] + [Tab]` もしくは `[左Shift] + [右Shift]`

デバッガ (debugger)


- プログラムのバグを見つける作業をサポートしてくれる便利なツール
- GDB (The GNU Debugger)
 - Unix で広く使われているデバッガ
 - 主な機能
 - ブレークポイント
 - ステップ実行
 - 変数値の表示
 - 関数の呼び出し
 - Emacs との連動

GDBの使い方(1)

- プログラムのコンパイル
 - デバッグ情報付きコンパイル
 - コンパイルする際に `-g` オプションをつける

```
% gcc -g life.c
```
- GDBの起動

```
% gdb a.out
```



デバッグしたいプログラム
- GDBの終了

```
(gdb) quit
```

GDBの使い方(2)

- ブレークポイントの設定
 - 例) 48行目で実行を止めて様子を見てみたい
 - (gdb) break 48
- プログラムの実行
 - (gdb) run
 - プログラムの実行が48行目(の直前)で中断する
- 変数の値の表示
 - 例) 変数 n の現時点での値を知りたい
 - (gdb) print n
- 変数の値の表示
 - 例) 変数 n の値を常に表示させておきたい
 - (gdb) display n

GDBの使い方(3)

- ステップ実行
 - この行を実行して次の行に進む
 - (gdb) next
- ステップ実行(関数の中に潜っていきたい場合)
 - (gdb) step
 - 実行する行に関数が含まれていない場合は next と同じ
- プログラムの実行の再開
 - (gdb) continue
 - 次のブレークポイントに遭遇するまでプログラムが実行される
- 指定行まで実行
 - 例) 51行目まで実行したい
 - (gdb) until 51
- 関数の呼び出し元の状態を知りたい
 - (gdb) up ← 一段上がる
 - (gdb) down ← 一段下がる

GDBの使い方(4)

- コマンドの省略形

コマンド	省略形
quit	q
break	b
run	r
print	p
display	disp
next	n
step	s
until	u

(その他便利な機能)
コマンドを何も入力せずに
リターンキーを押すと、直前
と同じコマンドが実行される

- Emacs との連携

- M-x gdb で起動
- ソースコード上で実行位置が表示される

今日の内容

- C言語入門
 - 構造体
 - 宣言、メンバ
 - 構造体を指すポインタ
 - 初期化
 - コマンドライン引数
- 物理シミュレーション
 - 多体問題

構造体

- 複数のデータ型をまとめ、新たなデータ型をつくる

例

学生	
学生証番号	整数
名前	文字列
年齢	整数
身長	浮動小数点数
体重	浮動小数点数

```
struct student
{
    int id;
    char name[100];
    int age;
    double height;
    double weight;
};
```

メンバ



構造体

- 構造体の宣言、定義、メンバへのアクセス

```
struct point
{
    int x;
    int y;
};

int main()
{
    struct point p1;

    p1.x = 10;
    p1.y = 20;
}
```

構造体

- 構造体の初期化・代入

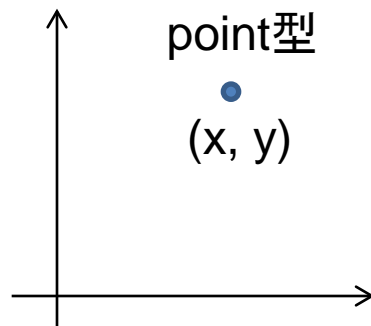
```
struct point
{
    int x;
    int y;
};
```

```
int main( )
{
    struct point a = { 10, 20 };
    struct point b;

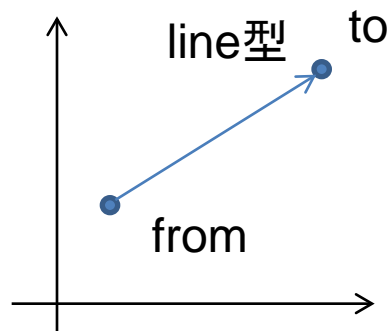
    b = a;
}
```

構造体

- 構造体をメンバに持つ構造体もつくれる



```
struct point
{
    int x;
    int y;
};
```



```
struct line
{
    struct point from;
    struct point to;
};
```

構造体

- 構造体を指すポインタ

```
struct point
{
    int x;
    int y;
};
```

```
int main()
{
    struct point *p = malloc(sizeof(struct point));

    (*p).x = 10;
    p->x = 10;

}
```

point 構造体ひとつぶんのメモリを確保
してその先頭アドレスを返す



どちらでも同じ

構造体

- 構造体へのポインタのよくある使い方
 - 関数呼び出し

```
void increment_age(struct student *s)
{
    s->age += 1;
}
```

↑
ポインタを受け取る

```
void some_function()
{
    struct student a = { 1, "Mike", 21, 175, 72 };

    increment_age(&a);
}
```

↖
構造体aが格納されているメモリの
アドレスを渡す


構造体

- もちろんポインタではなく値渡しでもできる

```
void print_age(struct student s)
{
    printf("age = %d\n", s.age);
}
```

- ただし、大きな構造体を渡すときには注意
 - 関数を呼び出すたびに構造体がコピーされるので無駄が大きい
- 大きな構造体を渡すときは原則としてポインタで

構造体の内容を変更しない場合は安全のため const をつける



```
void print_age(const struct student *s)
{
    printf("age = %d\n", s->age);
}
```


コマンドライン引数

- プログラムの実行時にパラメータやオプションを指定できる

`% ./a.out 0.1`

コレ



- 文字列へのポインタの配列として得られる
- 数値に変換したいときは `atoi`, `atof` 等を利用

(“a.out”も含めた)引数の数

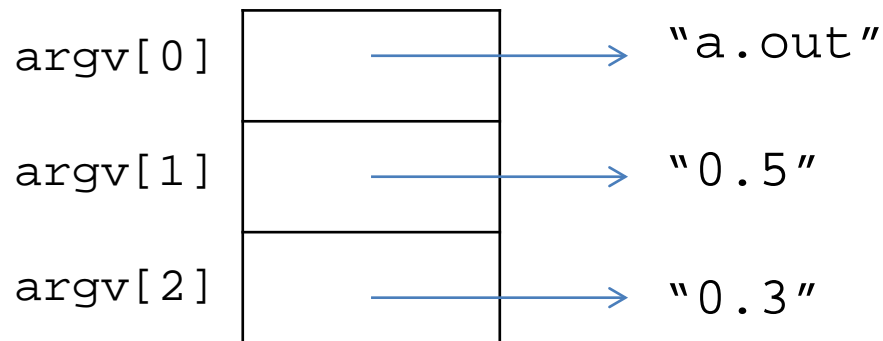


```
int main(int argc, char *argv[])
{
    int i;
    for (i = 0; i < argc; i++) {
        printf("%d %s\n", i, argv[i]);
    }
}
```

char *argv[] とは？

- 文字(列)へのポインタの配列

```
% ./a.out 0.5 0.3
```



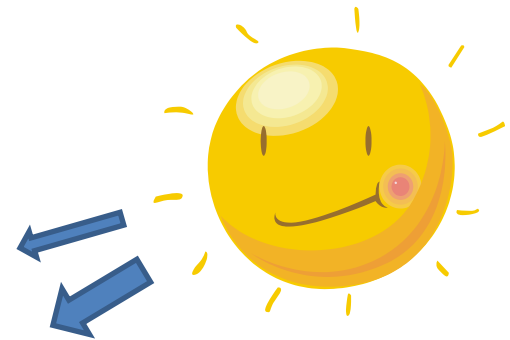
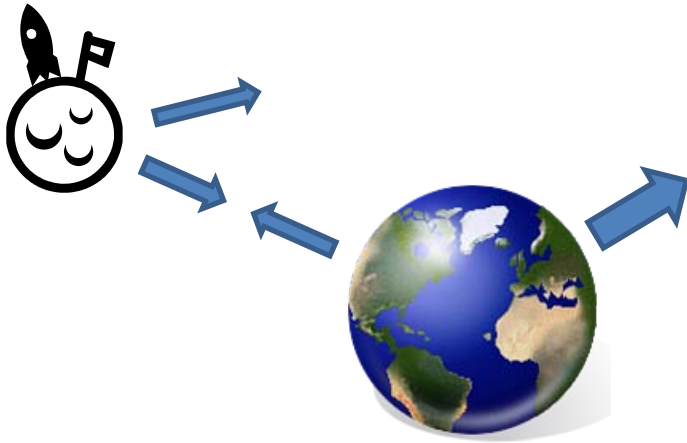
- よくある別な書き方

```
int main(int argc, char **argv)
```

↑
文字(列)へのポインタのポインタ

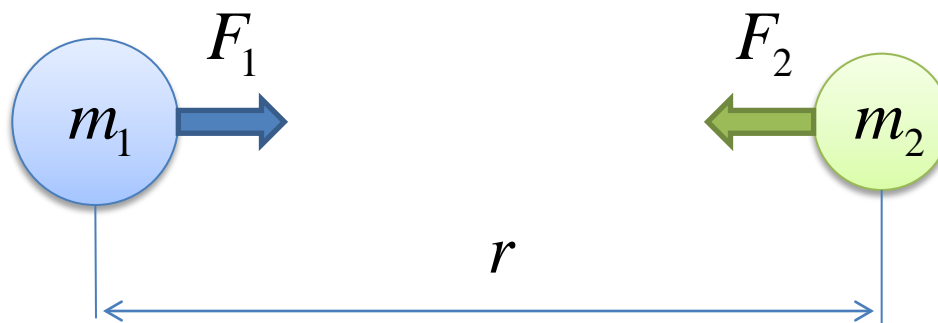
物理シミュレーション

- 多体問題 (n -body problem)
 - 万有引力による星の運動



万有引力

- 質量を有する2つの物体に働く引力



$$F_1 = F_2 = G \frac{m_1 m_2}{r^2}$$

重力定数 $G \approx 6.674 \times 10^{-11} [\text{m}^3 \text{s}^{-2} \text{kg}^{-1}]$

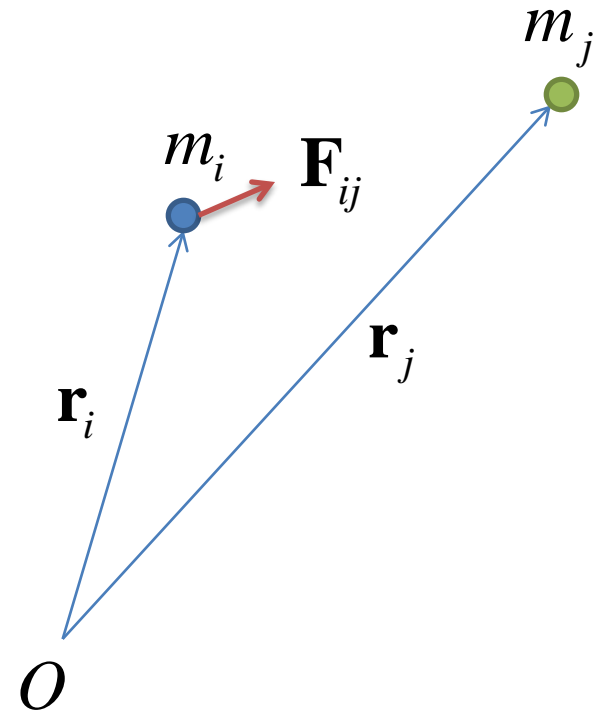
重力多体系

- 粒子 i が粒子 j から受ける重力(ベクトル)

$$\begin{aligned}\mathbf{F}_{ij} &= G \frac{m_i m_j}{|\mathbf{r}_j - \mathbf{r}_i|^2} \frac{\mathbf{r}_j - \mathbf{r}_i}{|\mathbf{r}_j - \mathbf{r}_i|} \\ &= G \frac{m_i m_j}{|\mathbf{r}_j - \mathbf{r}_i|^3} (\mathbf{r}_j - \mathbf{r}_i)\end{aligned}$$

- 多数の粒子から受ける重力

$$\mathbf{F}_i = \sum_{j \neq i} \mathbf{F}_{ij}$$



運動方程式

- 物体の運動を記述する微分方程式

位置

$$\frac{d\mathbf{r}_i}{dt} = \mathbf{v}_i$$

速度

$$m_i \frac{d\mathbf{v}_i}{dt} = \mathbf{F}_i$$

$$= Gm_i \sum_{j \neq i} \frac{m_j}{|\mathbf{r}_j - \mathbf{r}_i|^3} (\mathbf{r}_j - \mathbf{r}_i)$$



$$\frac{d\mathbf{v}_i}{dt} = G \sum_{j \neq i} \frac{m_j}{|\mathbf{r}_j - \mathbf{r}_i|^3} (\mathbf{r}_j - \mathbf{r}_i)$$

オイラー法 (Euler's method)

- 1階常微分方程式の数値解法

関数 $x(t)$ に関して以下が成り立つ

$$x'(t) = \lim_{\Delta t \rightarrow 0} \frac{x(t + \Delta t) - x(t)}{\Delta t}$$

$$x(t + \Delta t) \approx x(t) + x'(t)\Delta t$$



$$\left\{ \begin{aligned} \mathbf{a}_i^{(n)} &= G \sum_{j \neq i} \frac{m_j}{|\mathbf{r}_j^{(n)} - \mathbf{r}_i^{(n)}|^3} (\mathbf{r}_j^{(n)} - \mathbf{r}_i^{(n)}) \\ \mathbf{v}_i^{(n+1)} &= \mathbf{v}_i^{(n)} + \mathbf{a}_i^{(n)} \cdot \Delta t \\ \mathbf{r}_i^{(n+1)} &= \mathbf{r}_i^{(n)} + \mathbf{v}_i^{(n+1)} \cdot \Delta t \end{aligned} \right.$$

サンプルプログラム gravity.c

- コンパイル & 実行

```
% gcc -Wall gravity.c  
% ./a.out
```

- ターミナルをもうひとつ開く(表示用)

```
% tail -f space.txt
```

ターミナルのサイズをマウスで調整して ----- が左上にくるように

- 注意

- 何度も実行していると space.txt ファイルのサイズが大きくなる
ので適当なタイミングで消去すること

```
% rm space.txt
```


gravity.c 冒頭

- const 修飾子について

```
const double G = 1.0;
```

- 重力定数 G の値をグローバル変数として定義
- const をつけると変更が許されなくなる
 - バグ防止に有効
 - 変更しようとするコンパイル時にエラーになる

- コメントの形式

```
const double G = 1.0;    // gravity constant
```

```
const double G = 1.0;    /* gravity constant */
```

gravity.c 冒頭

- 構造体の宣言

```
struct star
{
    double m;    // mass
    double x;    // position_x
    double vx;   // velocity_x
};
```

- 構造体を使った配列の初期化

```
struct star stars[] = {
    { 1.0, -10.0, 0.0 },
    { 0.5,  10.0, 0.2 }
};
```

2つの星のデータを用意

重さ	x座標	速度(x成分)
1.0	-10.0	0
0.5	10.0	0.2

gravity.c 冒頭

- sizeof について
 - 変数、型、配列などの大きさバイトを単位として返す

```
const int nstars = sizeof(stars) / sizeof(struct star);
```

48バイト

24バイトの構造体が2つ
格納されているので

24バイト

double がひとつで8バイト
(64ビット)なので

- 割り算すると、結局 `nstars` が星の数になる

plot_stars()関数

- memset() について
 - 連続領域のメモリを指定したバイトで埋める

```
void plot_stars(FILE *fp, const double t)
{
    int i;
    char space[WIDTH][HEIGHT];

    memset(space, ' ', sizeof(space));
}
```

開始位置

&(space[0][0]) と等価

埋める内容

(空白文字)

バイト数

- char型の2次元配列を空白文字で初期化している
 - もちろん for 文の2重ループで初期化しても構わない

main()関数

- for 文の書き方について
 - 時間 t が `stop_time` に達するまで繰り返す
(と同時に、 i を 0 からカウントしていく)

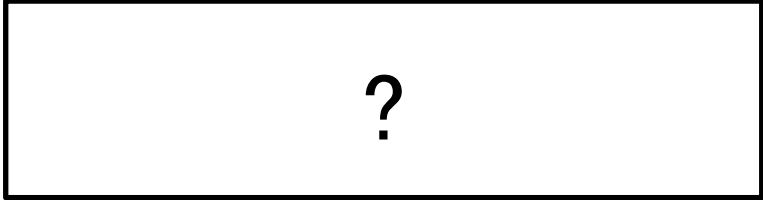
```
int i;  
double t;  
for (i = 0, t = 0; t <= stop_time; i++, t += dt)  
{  
    do_something();  
}
```

- 以下のように書いても同じ

```
int i = 0;  
double t = 0;  
while (t <= stop_time) {  
    do_something();  
    i++;  
    t += dt;  
}
```

update_velocities() 関数

- 星の速度を更新する

```
void update_velocities(const double dt)
{
    int i, j;
    for (i = 0; i < nstars; i++) {
        double ax = 0;
        
        stars[i].vx += ax * dt;
    }
}
```

他の星から受ける引力を
もとに加速度を計算



gravity.c その他

- usleep() 関数
 - スリープする時間をマイクロ秒で指定
- fabs() 関数
 - 浮動小数点数の絶対値を返す
 - update_velocities() 関数で使う

実習

1. `update_velocities` 関数を完成させる
2. 時間の刻み幅 Δt をコマンドライン引数で指定できるようにする

レポート課題(締め切り12/2)

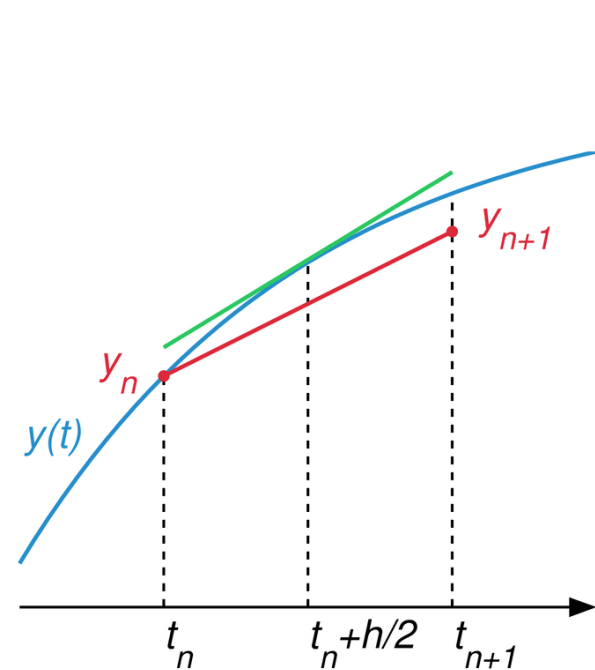
1. 2次元空間を扱えるように拡張せよ
 - 星の数を3つ以上に増やして動作を確認すること
 - プログラムを添付すること(ファイル名は“gravity1.c”)
2. 星と星との衝突(融合)現象を実装せよ
 - 星同士の距離がある値以内になったら融合するとしてよい(運動量保存)
 - プログラムを添付すること(ファイル名は“gravity2.c”)
3. [発展課題] 本アプリケーションをさらに発展させよ
 - プログラムを添付すること(ファイル名は“gravity3.c”)
 - 発展のさせ方は任意
 - 例) 中点法等によるシミュレーションの精度向上、3次元空間に拡張、地球と月と太陽の配置を再現してみる、衝突による破壊・散乱を考慮、電界・磁界を考慮、etc.
 - どのように発展させたのか簡潔に説明すること

課題の提出方法

- 宛先
 - software2@logos.t.u-tokyo.ac.jp
- Subject
 - 形式: SOFT-MM-DD-NNNNNNNX
 - MM: 月
 - DD: 日 (授業が行われた日)
 - NNNNNNX: 学籍番号
- 本文
 - 冒頭に学籍番号、氏名を明記

参考 中点法 (Midpoint method)

- 2次のルンゲ・クッタ法 (Runge-Kutta method)



http://en.wikipedia.org/wiki/Midpoint_method

$$\mathbf{r}_i^{(n+\frac{1}{2})} = \mathbf{r}_i^{(n)} + \mathbf{v}_i^{(n)} \cdot \frac{\Delta t}{2}$$

$$\mathbf{a}_i^{(n+\frac{1}{2})} = G \sum_{j \neq i} \frac{m_j}{\left| \mathbf{r}_j^{(n+\frac{1}{2})} - \mathbf{r}_i^{(n+\frac{1}{2})} \right|^3} \left(\mathbf{r}_j^{(n+\frac{1}{2})} - \mathbf{r}_i^{(n+\frac{1}{2})} \right)$$

$$\mathbf{v}_i^{(n+1)} = \mathbf{v}_i^{(n)} + \mathbf{a}_i^{(n+\frac{1}{2})} \cdot \Delta t$$

$$\mathbf{r}_i^{(n+1)} = \mathbf{r}_i^{(n)} + \frac{(\mathbf{v}_i^{(n+1)} + \mathbf{v}_i^{(n)})}{2} \cdot \Delta t$$

精度向上に関して拡張する場合は、精度が向上したことがわかる実例も記述すること
(理論値とのずれが小さくなった、 Δt を小さくするのと同じ効果が得られた、など)