生態模擬： 以 C 語言為例

Class 09 (2018/05/24)

Applications of Pointer and Array

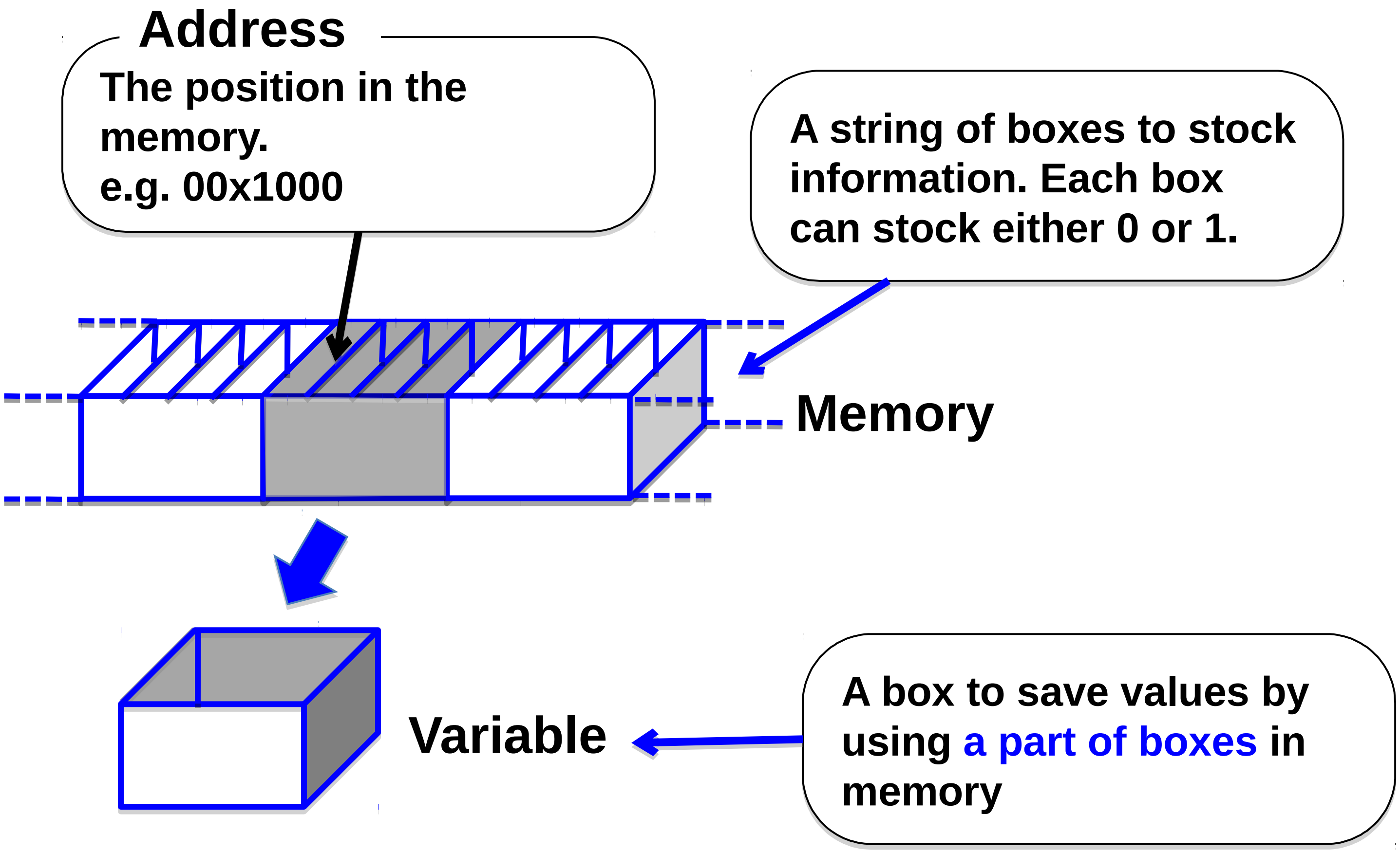
- 9.1 Review of address, pointer, and array
- 9.2 Array and pointer as parameter in function
- 9.3 Pointer to function (function pointer)
- 9.4 4^{th}-order Runge-Kutta using function pointer
- 9.5 Homework for using Pointer to function

Takeshi Miki

三木　健（海洋研究所）

Memory, Address, and Variable (review)

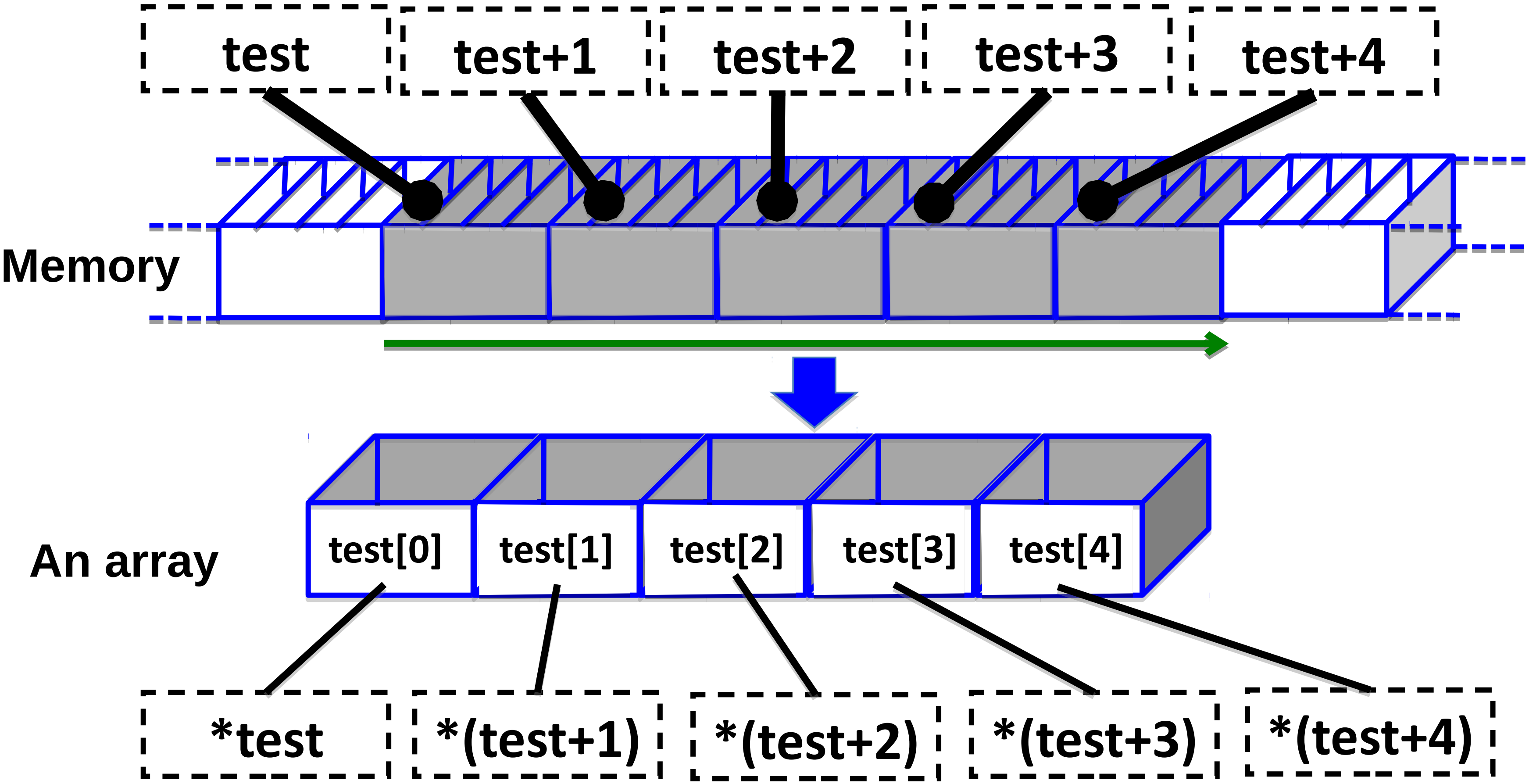
The address in C represents the <u>position</u> in the <u>memory</u>, which is occupied by a <u>variable</u>.



1 Array and Pointer (review)

The pointer operators (*, +, -).

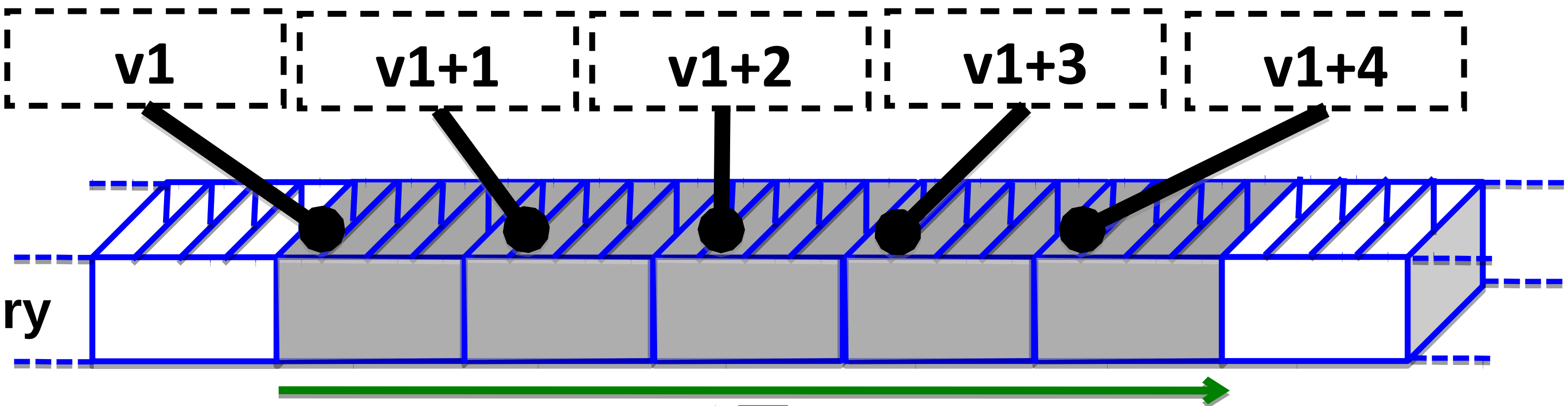
double test[5];



namic allocation of memory to array via pointer (revi

The pointer operators (*, +, -) and subscript operator ([])

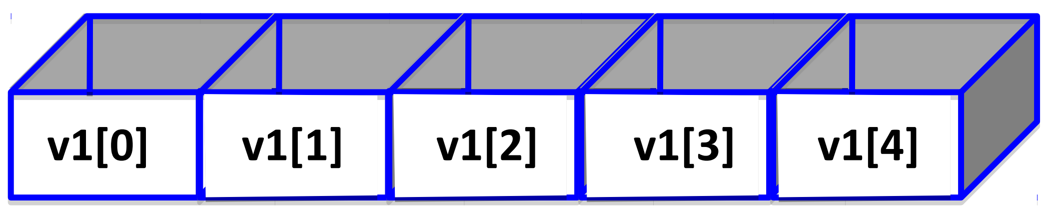
```
double *v1;
v1 = d_vector(4);
```



Memory

```
v1 = (double *) malloc( (size_t) ((4 + 1)*sizeof(double)));
```

The array v1



v1[i] == *(v1 + i)

Array and pointer as parameter in function

Array and pointer can be parameter of function. The following 3 functions act in the same way.

```
double avg1(int t[])
{
    int j;
    double sum = 0.0;
    for (j=0; j < 5; j++) sum += t[j];
    return sum/5.0;
}
```

You can call this function in main () using the array name as argument.

```
int test[5];
avg1(test);
arg2(test);
arg3(test);
```

```
double avg2(int *pT)
{
    int j;
    double sum = 0.0;
    for (j=0; j < 5; j++) sum += *(pT + j);
    return sum/5.0;
}
```

```
double avg3(int *pT)
{
    int j;
    double sum = 0.0;
    for (j=0; j < 5; j++) sum += pT[j];
    return sum/5.0;
}
```

Pointer to function (function pointer)

How to declare a *pointer variable* to function (= function pointer).

構文 (Syntax):

*Data type of return value (*name of function pointer) (parameter list)*;

```
int (*pM) (int x, int y);
```

How to use it (1)?

```
int sum(int x, int y);

int main(void)
{
    int num1, num2, num3;
    int (*pM)(int x, int y);          //declaration of function pointer
    pM = sum;                         //assignment of address of sum
    num3 = (*pM) (num1, num2);        //call the assigned function: sum
    ....
```

Pointer to function (function pointer)

How to use it (2): more useful way

```
int sum(int x, int y);
int prod(int x, int y);

int main(void)
{
    int num1, num2, num3;
    int (*pM)[2](int x, int y);     //declaration of array of function pointer
    pM[0] = sum;                    //assignment of address of sum
    pM[1] = prod;                   //assignment of address of prod
    printf("Do you want to calculate summation (0) or product (1) of 3 & 6?\n");
    scanf("%d", &num1);
    num2 = (*pM[num1])(3, 6); //call of function using pointer
    printf("Calculated value is %d.\n", num2);

    return 0;
}
```

Pointer to function (function pointer)

How to use it (3): more useful way in numerical calculations

Function pointer can be parameter of function!!

rk4(double y[], …, void (*diff) (double in[], double out[]));

$$\begin{cases} \dfrac{dx}{dt} = r \cdot x\left(1 - \dfrac{x}{K}\right) - a \cdot x \cdot y \\ \dfrac{dy}{dt} = b \cdot x \cdot y - m \cdot y \\ \\ x(0) = 0.1, y(0) = 0.1 \end{cases}$$

Our target!!

4-th order Runge-Kutta method for ODE

Runge-Kutta method is the same for the multidimensional ODE.

$$\begin{cases} \dfrac{d\vec{N}(t)}{dt} = \vec{f}\left(\vec{N}(t), t\right) \\ \vec{N}(0) = \vec{N}_0 \end{cases}$$

(1) We need to define function to calculate *f* value.

The 4th order (explicit) Runge-Kutta method is..

$$\vec{n}_{i+1} = \vec{n}_i + h \cdot \frac{1}{6}(\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4)$$

with

$$\begin{cases} \vec{k}_1 = \vec{f}(\vec{n}_i, t_i) \\ \vec{k}_2 = \vec{f}\left(\vec{n}_i + \dfrac{h}{2}\vec{k}_1, t_i + \dfrac{h}{2}\right) \\ \vec{k}_3 = \vec{f}\left(\vec{n}_i + \dfrac{h}{2}\vec{k}_2, t_i + \dfrac{h}{2}\right) \\ \vec{k}_4 = \vec{f}(\vec{n}_i + h\vec{k}_3, t_i + h) \end{cases}$$

(2) We need to define function for calculating 1-step of Runge-Kutta.

1) We need to define function to calculate *f* value.

```
void differential(double time, double in[], double out[]);
double dx_dt(double time, double vr[]);
double dy_dt(double time, double vr[]);

int j_x = 1;   //define index as global variable
int j_y = 2;   //define index as global variable
double a = 1.0;  //define model parameter as global
...
int main(void)
{
     ....
}

void differential(double time, double in[], double out[]);
{
     out[j_x] = dx_dt(time, in);
     out[j_y] = dy_dt(time, in);
}
double dx_dt(double time, double vr[])
{
    return r*(1.0 – vr[j_x]/K) -  a*vr[j_x]*vr[j_y];
}
```

```
int main(void)
{
       double t = 0.0;
       double deltat = 1.0e-3;

       double *v = d_vector(2);
       double *dfdt = d_vector(2);
       ...
       v[j_x] = 1.0; //initial density
       v[j_y] = 2.0; //initial density
       ...

       differential(t, v, dfdt);  ]
       rk4(...);                  ] 1-step
       t += deltat;               ]

       ....
       free_d_vector(v);
       free_d_vector(dfdt);
       return 0;
}
```

We need to define function for calculating 1-step of Runge-Kutt

```
void rk4(double y[], double dydt[], int n, double t, double h, double yout[],
void (*diff) (double, double [], double []))
{
        int i;
        double th, hh, h6, *dym, *dyn, *dyt, *yt;

        dym = d_vector(n);
        dyn = d_vector(n);
        dyt =d_vector(n);
        yt = d_vector(n);
        hh = h*0.5;
        h6 = h/6.0;
        th = t + hh;
        for (i=1;i<=n;i++) yt[i] = y[i] + hh*dydt[i];
        (*diff)(th, yt, dyt);
        for (i=1;i<=n;i++) yt[i] = y[i] + hh*dyt[i];
        (*diff)(th, yt, dym);
        for (i=1;i<=n;i++) yt[i] = y[i] + h*dym[i];
        (*diff)(t+h, yt, dyn);

        for (i=1;i<=n;i++)  yout[i] = y[i] + h6*(dydt[i]+2.0*dyt[i]+2.0*dym[i] +
dyn[i]);

        free_d_vector(yt);
        free_d_vector(dyt);
        free_d_vector(dym);
        free_d_vector(dyn);
}
```

$$\vec{k}_1 = \vec{f}(\vec{n}_i, t_i)$$ **(already calculated)**

$$\vec{n}_i + \frac{h}{2}\vec{k}_1$$

$$\vec{k}_2 = \vec{f}\left(\vec{n}_i + \frac{h}{2}\vec{k}_1, t_i + \frac{h}{2}\right)$$

$$\vec{n}_i + \frac{h}{2}\vec{k}_2$$

$$\vec{k}_3 = \vec{f}\left(\vec{n}_i + \frac{h}{2}\vec{k}_2, t_i + \frac{h}{2}\right)$$

$$\vec{n}_i + h\vec{k}_3$$

$$\vec{k}_4 = \vec{f}(\vec{n}_i + h\vec{k}_3, t_i + h)$$

```
int main(void)
{
      ...
      differential(t, v, dfdt);
      rk4(v, dfdt, 2, t, deltat, v,
differential);
      t += deltat;
     ....
}
```

Application to a prey-predator model

Numerically solve the following equations with the Runge-Kutta method.

$$
\begin{cases}
\dfrac{dx}{dt} = r \cdot x\left(1 - \dfrac{x}{K}\right) - a \cdot x \cdot y \\[2ex]
\dfrac{dy}{dt} = b \cdot x \cdot y - m \cdot y \\[6ex]
x(0) = 0.1, y(0) = 0.1
\end{cases}
$$

Our target!!

$$r = 1.0, K = 1, a = ?, b = ?, m = ?$$

Application to a prey-predator model

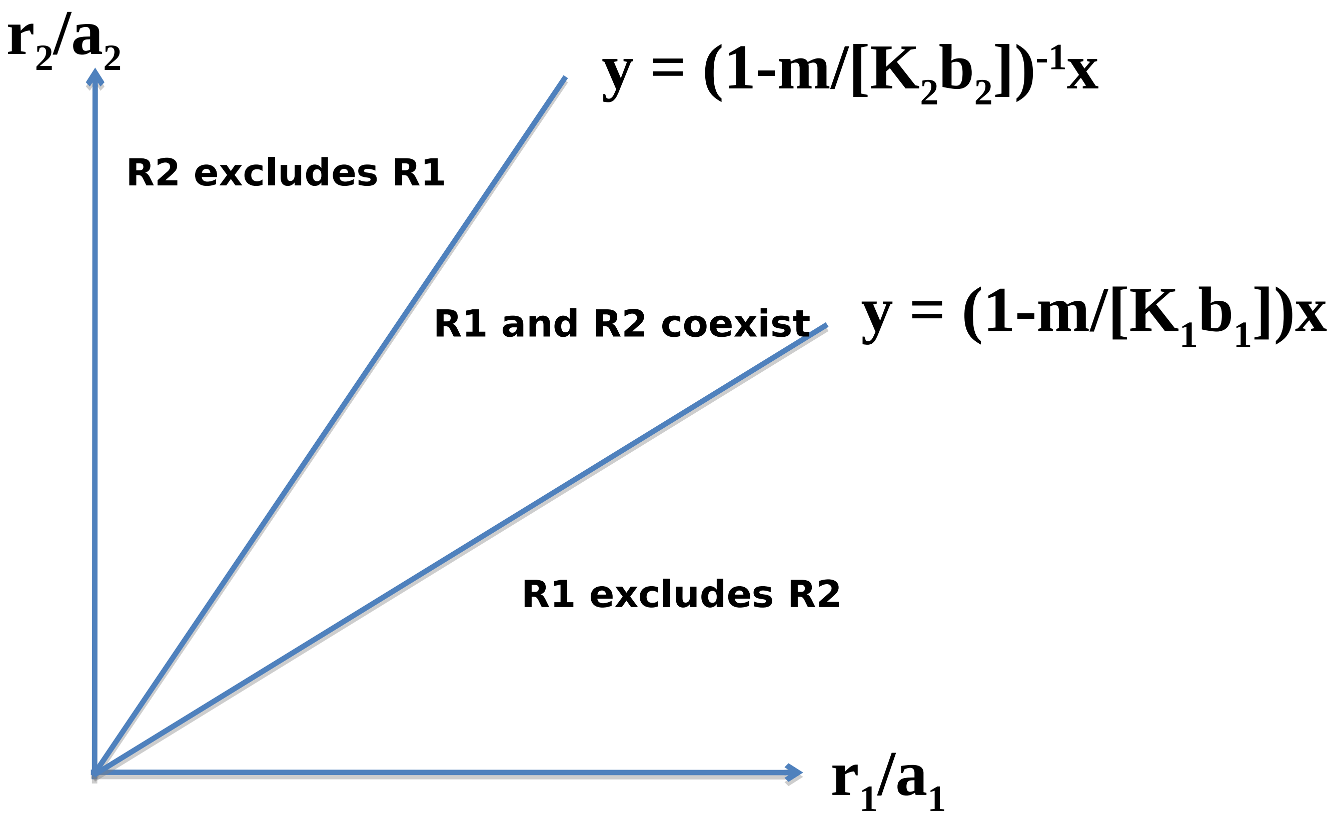
Numerically solve the following equations (an apparent competition) with the Runge-Kutta method and show the temporal dynamics of them in three cases (R1 excludes R2, R2 excludes R1, and R1 and R2 coexist).

$$
\begin{cases}
\dfrac{dR_1}{dt} = \left\{ r_1 \left(1 - \dfrac{R_1}{K_1} \right) - a_1 P \right\} R_1 \\[2ex]
\dfrac{dR_2}{dt} = \left\{ r_2 \left(1 - \dfrac{R_2}{K_2} \right) - a_2 P \right\} R_2 \\[2ex]
\dfrac{dP}{dt} = \left(b_1 R_1 + b_2 R_2 - m \right) P \\[2ex]
R_1(0) > 0, R_2(0) > 0, P(0) > 0
\end{cases}
$$

Homework !

Application to a prey-predator model

Numerically solve the following equations (an apparent competition) with the Runge-Kutta method and show the temporal dynamics of them in three cases (R1 excludes R2, R2 excludes R1, and R1 and R2 coexist).



Homework !