과제 설명

본 과제는 초기값 문제

$$\frac{dy}{dt} = f(t,y) = y - t^2 + 1,\ y(0) = 0.5$$

의 해를 Euler, Modified-Euler, RK4 방법을 이용해 수치적으로 구하는 과제이다
위 세 방법 모두 구할 구간을 N 개의 스텝으로 나눠 첫 지점 부터 한 스텝씩 반복적으로 구하는 방법이다
초기값 문제의 정확한 해는 $y(t) = (t+1)^2 - 0.5e^t$ 이며 수치적으로 구한 값과 정확 해를 표로 나타내 비교할 수 있게 함이 목표이다

Euler Method

$$w_{i+1} = w_i + hf(t_i, w_i) \qquad (식\ 1)$$

현재 함수값에 (현 시점에 주어진 미분 f) * (스텝 크기)를 더해 다음 스텝의 함수값을 근사하는 기법이다
테일러 1차 전개를 사용한 효과를 낸다

Modified-Euler Method

$$w_{i+1} = w_i + \frac{h}{2}[f(t_i, w_i) + f(t_{i+1}, w_i + hf(t_i, w_i))] \qquad (식\ 2)$$
$$\rightarrow k_1 = f(t_i, w_i),\ k_2 = f(t_{i+1}, w_i + hk_1)$$
$$w_{i+1} = w_i + \frac{h}{2}[k_1 + k_2]$$

(현 시점에 주어진 미분), (오일러 방법으로 예측한 다음 점에 주어진 미분) 을 평균내어 사용할 기울기를 정한 후,
(스텝 크기)를 곱해 현재 함수값에 더하여 다음 스텝의 함수값을 근사하는 기법이다
테일러 2차 전개를 사용한 효과를 낸다
구현 편의를 위해 중간변수 k_1, k_2 를 사용하였다

RK4 Method

$$k_1 = hf(t_i, w_i),\ k_2 = hf(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_1) \qquad (식\ 3)$$
$$k_3 = hf(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_2),\ k_4 = hf(t_{i+1}, w_i + k_3)$$
$$w_{i+1} = w_i + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4]$$

변수	의미
k_1	현 상태에 주어진 기울기로 추정한 증분
k_2	k_1 로 반 스텝 예측한 상태에 주어진 기울기로 추정한 증분
k_3	k_2 로 반 스텝 예측한 상태에 주어진 기울기로 추정한 증분
k_4	k_3 로 한 스텝 예측한 상태에 주어진 기울기로 추정한 증분
w_{i+1}	k_i 들을 가중평균하여 예측한 한 스텝 뒤의 값

테일러 4차 전개를 사용한 효과를 낸다

코드 구현법

`double f(double t, double y)`

문제로 주어진 $f(t,y)$를 구현한다

`typedef`를 이용해 위 함수의 함수 포인터 타입을 `func_t`라고 정의하였다

`double exact_f(double t)`

위 초기값 문제의 정확한 해인 $y(t)$ 를 구현한다

`double euler_step(double w, double h, func_t f, double t)`

현 시점의 t, w, 스텝사이즈 h, 주어진 미분 f 를 받아 다음 시점의 함수값 w_{i+1} 을 **Euler Method** 로 구한 후 반환한다. 식 (1)을 그대로 구현하였다

`double M_euler_step(double w, double h, func_t f, double t)`

현 시점의 t, w, 스텝사이즈 h, 주어진 미분 f 를 받아 다음 시점의 함수값 w_{i+1} 을 **Modified-Euler Method** 로 구한 후 반환한다. 식 (2)을 그대로 구현하였다

`double RK4_step(double w, double h, func_t f, double t)`

현 시점의 t, w, 스텝사이즈 h, 주어진 미분 f 를 받아 다음 시점의 함수값 w_{i+1} 을 **RK4 Method** 로 구한 후 반환한다. 식 (3)을 그대로 구현하였다

`main`

	의미
`exact`	현 시점 정확한 해 $y(t)$ 가 담김
`w_euler`	Euler Method로 구한 값이 담김
`w_M_euler`	Modified-Euler Method로 구한 값이 담김
`w_rk4`	RK4로 구한 값이 담김
초기 조건	`w_euler = w_M_euler = w_rk4 = 0.5` $(y(0)=0.5)$
스텝 사이즈	`h_euler=0.025`, `h_M_euler=0.05`, `h_rk4=0.1`
`step`	루프인덱스, 본 과제에선 $t=0$ 부터 $t=2$ 까지 0.1 간격으로 값을 출력 하므로 루프는 21 번 돌아야한다 따라서 `step`은 0 부터 20 까지 반복된다

메인 루프

각 반복에서 수행 내용

1. **`step`에 따른 `t` 계산** : 본 과제에서는 0.1 단위로 출력하므로 `t = step * 0.1`
2. **정확해 계산**: `exact = exact_f(t)`
3. **현재 시점의 계산값 출력** : `exact, w_euler, w_M_euler, w_rk4` 를 **시점 `t`** 에 맞춰 한 줄 출력
   `step=0`에서는 초기조건 $t=0$ 에서의 값이 출력된다
   이전 반복 끝에서 이미 다음 시점으로 업데이트했으므로, **출력, 계산** 순서가 자연스럽게 맞물린다
4. **다음 출력 값 계산** : 출력 간격이 0.1 이므로, 각 반복의 끝에 **Euler**는 4 회($h=0.025$), **Modified-Euler**는 2 회($h=0.05$), **RK4**는 1 회($h=0.1$) 계산을 수행한다.
   이렇게 하면 **다음 반복에서 출력되는 값**이 정확히 `t + 0.1 (= 0.1 * (step + 1))` 시점에 대응한다. 구현에서는 `for` 문으로 Euler 4회, Modified-Euler 2회를 수행했다.

```c
//전체 소스코드
#include <stdio.h>
#include <math.h>
typedef double (*func_t)(double, double);

double f(double t, double y) {
    return y - t * t + 1;
}
double exact_f(double t) {
    return (t + 1) * (t + 1) - 0.5 * exp(t);
}

double euler_step(double w, double h, func_t f, double t) {
    return w + h * f(t, w);
}
double M_euler_step(double w, double h, func_t f, double t) {
    double k1 = f(t, w);
    double k2 = f(t + h, w + h * k1);
    return w + h / 2.0 * (k1 + k2);
}
double RK4_step(double w, double h, func_t f, double t) {
    double k1 = h * f(t, w);
    double k2 = h * f(t + h / 2, w + k1 / 2);
    double k3 = h * f(t + h / 2, w + k2 / 2);
    double k4 = h * f(t + h, w + k3);
    return w + 1.0 / 6.0 * (k1 + 2 * k2 + 2 * k3 + k4);
}

int main()
{
    double exact = 0.5;
    double w_euler = 0.5, w_M_euler = 0.5, w_rk4 = 0.5;
    double h_euler = 0.025, h_M_euler = 0.05, h_rk4 = 0.1;

    printf("                Euler      M-Euler    RK4\n");
    printf("t    Exact      h=0.025    h=0.05     h=0.1\n");

    for (int step = 0; step < 21; step += 1) {
        double t = step * 0.1;
        exact = exact_f(step * 0.1);

        printf("%.1f  %.7f  %.7f  %.7f  %.7f\n",
                t, exact, w_euler, w_M_euler, w_rk4);

        for (int i = 0; i < 4; i += 1)
            w_euler = euler_step(w_euler, h_euler, f, t + i * h_euler);
        for (int i = 0; i < 2; i += 1)
            w_M_euler = M_euler_step(w_M_euler, h_M_euler, f, t + i * h_M_euler);
        w_rk4 = RK4_step(w_rk4, h_rk4, f, t);
    }
    return 0;
}
```

실행결과

```
                Euler       M-Euler     RK4
t     Exact     h=0.025     h=0.05      h=0.1
0.0   0.5000000 0.5000000   0.5000000   0.5000000
0.1   0.6574145 0.6554982   0.6573085   0.6574144
0.2   0.8292986 0.8253385   0.8290778   0.8292983
0.3   1.0150706 1.0089334   1.0147254   1.0150701
0.4   1.2140877 1.2056345   1.2136079   1.2140869
0.5   1.4256394 1.4147264   1.4250141   1.4256384
0.6   1.6489406 1.6354189   1.6481579   1.6489394
0.7   1.8831236 1.8668401   1.8821709   1.8831222
0.8   2.1272295 2.1080276   2.1260931   2.1272278
0.9   2.3801984 2.3579190   2.3788637   2.3801964
1.0   2.6408591 2.6153415   2.6393103   2.6408567
1.1   2.9079170 2.8790008   2.9061375   2.9079143
1.2   3.1799415 3.1474680   3.1779134   3.1799385
1.3   3.4553517 3.4191660   3.4530558   3.4553482
1.4   3.7324000 3.6923540   3.7298159   3.7323961
1.5   4.0091555 3.9651104   4.0062614   4.0091511
1.6   4.2834838 4.2353141   4.2802567   4.2834790
1.7   4.5530263 4.5006238   4.5494416   4.5530210
1.8   4.8151763 4.7584553   4.8112079   4.8151704
1.9   5.0670528 5.0059559   5.0626732   5.0670464
2.0   5.3054720 5.2399769   5.3006521   5.3054650
```

t_i	Exact	Euler $h = 0.025$	Modified Euler $h = 0.05$	Runge-Kutta Order 4 $h = 0.1$
0.0	0.5000000	0.5000000	0.5000000	0.5000000
0.1	0.6574145	0.6554982	0.6573085	0.6574144
0.2	0.8292986	0.8253385	0.8290778	0.8292983
0.3	1.0150706	1.0089334	1.0147254	1.0150701
0.4	1.2140877	1.2056345	1.2136079	1.2140869
0.5	1.4256394	1.4147264	1.4250141	1.4256384

실행 결과 표 `Table.5.8`

본 프로그램이 표 `Table.5.8`를 완벽히 재구성함을 알 수 있다